LBID 1575

Anat GRYNBERG
July 1989

Validation of Radiance

Lawrence Berkeley Laboratory
Applied Science Division
Lighting Systems Research Group

TABLE OF CONTENTS

Validation Of Radiance

Introduction

Traditionally, lighting designers and architects have designed using only their own experience and assumptions. In increasing numbers, they are turning to CAD systems and design tools. **Radiance** is a design analysis tool for electric lighting and daylighting. The program was conceived as a research tool to study illumination, and has been used to prepare general guidelines and instruct designers in qualitative and quantitative aspects of lighting. The first part of this report will show the importance of lighting simulation in designing for energy efficiency, visual comfort and aesthetics. Radiance takes a scene description with light sources, sun, sky, other buildings, rooms, furniture, etc. and produces spectral radiance values which can be collected in a color image. Its specialty is providing both *realistic images* and *numerical data*. Designers can see what their work will look like and simultaneously, use the numerical data (luminance or illuminance values) to check the desired amount of light, or to compare the values with those given by other light sources.

Radiance uses a backwards ray-tracing luminance computation that follows light from the point of measurement to the source(s), taking into account specular reflection, transmission, and virtually any geometry. In the appendix are two papers concerning the technique used by the program.

In order to make use of numerical data and of computed pictures, one needs to know their accuracy. In other words, one needs to validate the program to prove its accuracy so that users have confidence in its output.

Last year we conducted a validation based on the comparison of Radiance with another lighting simulation program, Superlite, and with the results of measurements of physical scale models. Because Superlite can only handle diffuse surfaces, whereas Radiance can handle any kind of materials (specular, semi-specular and diffuse), the validation concerns just a small part of the Radiance program.

The results of the first validation were very encouraging. Radiance gave good results in all cases. We decided to generalize this validation. Especially, we wanted to build a reflectometer in order to measure the reflective characteristics of materials and to compare them to the Radiance simulation. We also wanted to conduct measurements on and simulate an *existing space* that has both electric light and daylight, in order to compare results of real world complexity. The second part of this report will present the validation we conducted.

First we shall show why and how Radiance can really be useful for the lighting design. To demonstrate this we chose an example, a simple rectangular office space to which we will add partitions and furniture to mimic a progressive design process (I). This will show the importance of model detail in the prediction of light levels and visibility. The second part of this report (II) will concern the validation itself including

1) the modeling of an existing place, a conference room, and the comparison of this scene between real and computed images.

2) the description of a reflectometer's prototype which should allow measurement of the reflectance of any material.

A third part (III) will show how Radiance can be applied to a real project. We will describe the modeling of a museum, the analysis of the results, and how the lighting designer and the architects will use this information.

I. Radiance used for lighting design - Office Space Example.

The following is an excerpt from the publication "*Luminance in Computer-Aided Lighting Design*" which was presented at the Vancouver Building Simulation conference in June 1989 [Ward89.6].

"The room is 92 by 51 foot rectangular and has a 10 foot ceiling. The diffuse reflectances of the ceiling, walls and floor are 80%, 50% and 25% respectively. Seventy-two 2 by 4 foot two-lamp lensed fluorescent fixtures were arranged on a uniform grid with 7 rows and 12 columns designed to provide an illuminance level around 750 (maintained) lux at the workplane. The number of fixtures required to deliver this level was calculated using the standard IES room cavity ratio method [Kaufman81]. The initial layout and computed light levels along the length of the space are shown in Figure 1.

Two rows of 16 paired cubicle partitions each were then added to the model space as shown in Figure 2. The average and standard deviation of illuminance at each desk in the second row of cubicles were calculated from 18 points arranged in two evenly distributed rows on each desk. Note that the partitions cause the light levels at the desk surface to drop by about a factor of two compared with the empty space, and that uniformity is severely compromised. The effect of partitions on workplane illuminance has been investigated empirically [Siminovitch87] and can be approximated analytically [Ballman87]. However, the analysis of partitions is not practiced widely due to the lack of computational tools for accurately assessing their effect. Using a general luminance calculation, all obstructions (not just rectilinear partitions) can be considered, requiring only a few minutes to compute.

To create a more uniform distribution of light on the cubicle desks, a second luminaire layout was designed with the fixtures directly over the partition walls as shown in Figure 3. The graph shows the new average and standard deviation of illuminance on the desks in the second row of partition cubicles. With this arrangement, each cubicle has two halves of a fixture on each side of its desk, which tends to minimize body shadowing and veiling reflection and produce high uniformity. To reduce the lighting of the central aisle between the partition rows, single lamp fixtures were used on either side. Four fixtures were used at the end corridors to provide acceptable lighting on the perimeter. The total lamp count for this arrangement is 96, down from 168 for the first design -- **an energy savings of 43%.** The graphs in Figure 4 show illuminance values at 18 sample locations on a representative desk under the two lighting conditions. The light levels of the nine sensors in the row furthest from the occupant are nearly as high as those in the near row.

The real power of a luminance simulation comes into play when the user demands a more intuitive representation of the light distribution on the desks. Figure 5 compares luminance maps (images) of the same desk under each fixture arrangement. These images give the shadows and brightness ranges in full detail, eliminating the need for guesswork. In an interactive display, individual luminance values can be extracted from the finished calculation at selected image locations. This combination of numerical and visual information is extremely useful to a lighting designer, comparable to taking a light meter into a finished space -- without first having to build it.

By adding recognizable objects to the model, further detail comes out of the calculation, such as the effect of light distribution on task visibility. Figure 6 is the same as Figure 5 with books and other miscellany added to the model. Suddenly the images take on real meaning, and unquantifiable considerations such as *visual quality* and atmosphere are accessible from what is still a computer calculation. With a simple change in view, one can evaluate the visibility of a reading task in this environment (Figure 7), and the dependence on view angle and surface specularity is shown by the simulation in a form that is immediately perceptible.

Task lighting will often improve visibility substantially for a modest investment. A complementary pair of compact fluorescent fixtures was added at each desk to show the effect of one type of task lighting. Figure 8 shows the light distribution when the ambient light is provided by the second overhead layout.

This example demonstrates some important lighting issues that can be addressed in a luminance calculation, such as uniformity, shadowing, glare and visibility. At each level of detail, new information is produced by the simulation, allowing the designer to optimize his lighting solution without resorting to physical models or uncertain assumptions. Although some of these questions could be answered by conventional calculations, the work involved is daunting and the results are less intuitive."

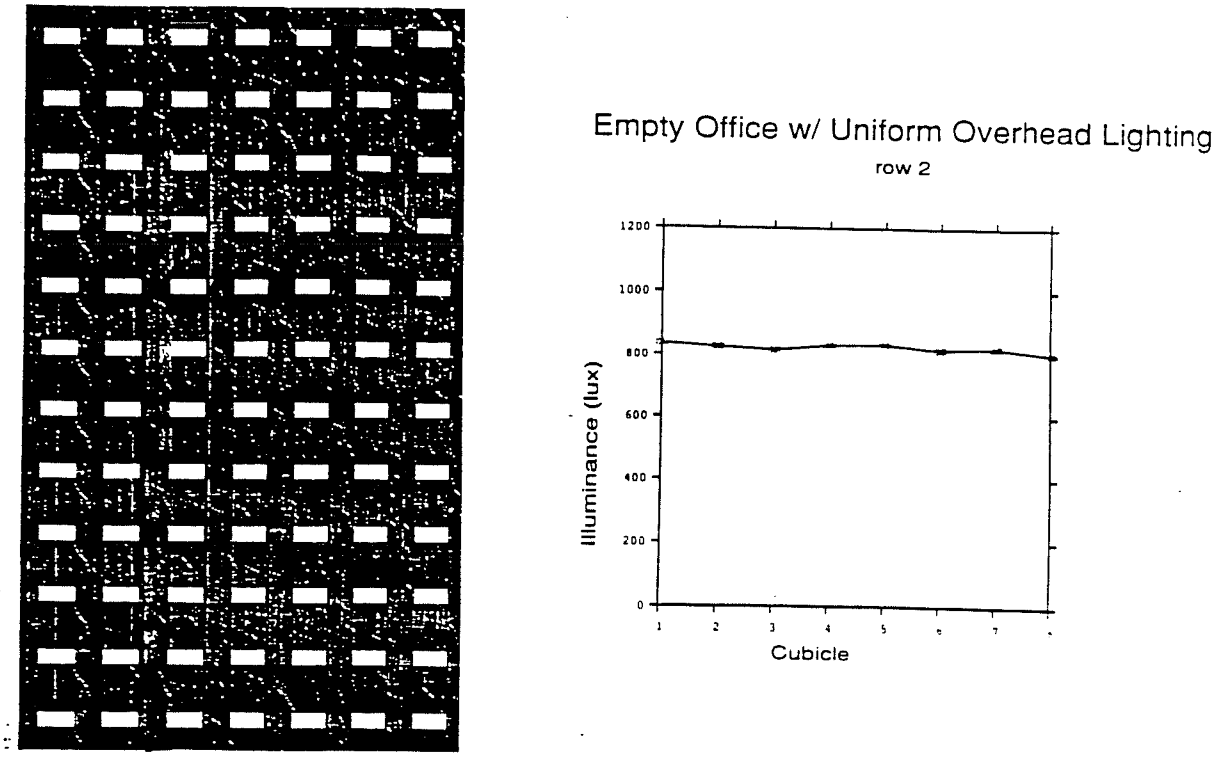


Figure 1. Initial luminaire layout for office space and corresponding light levels.

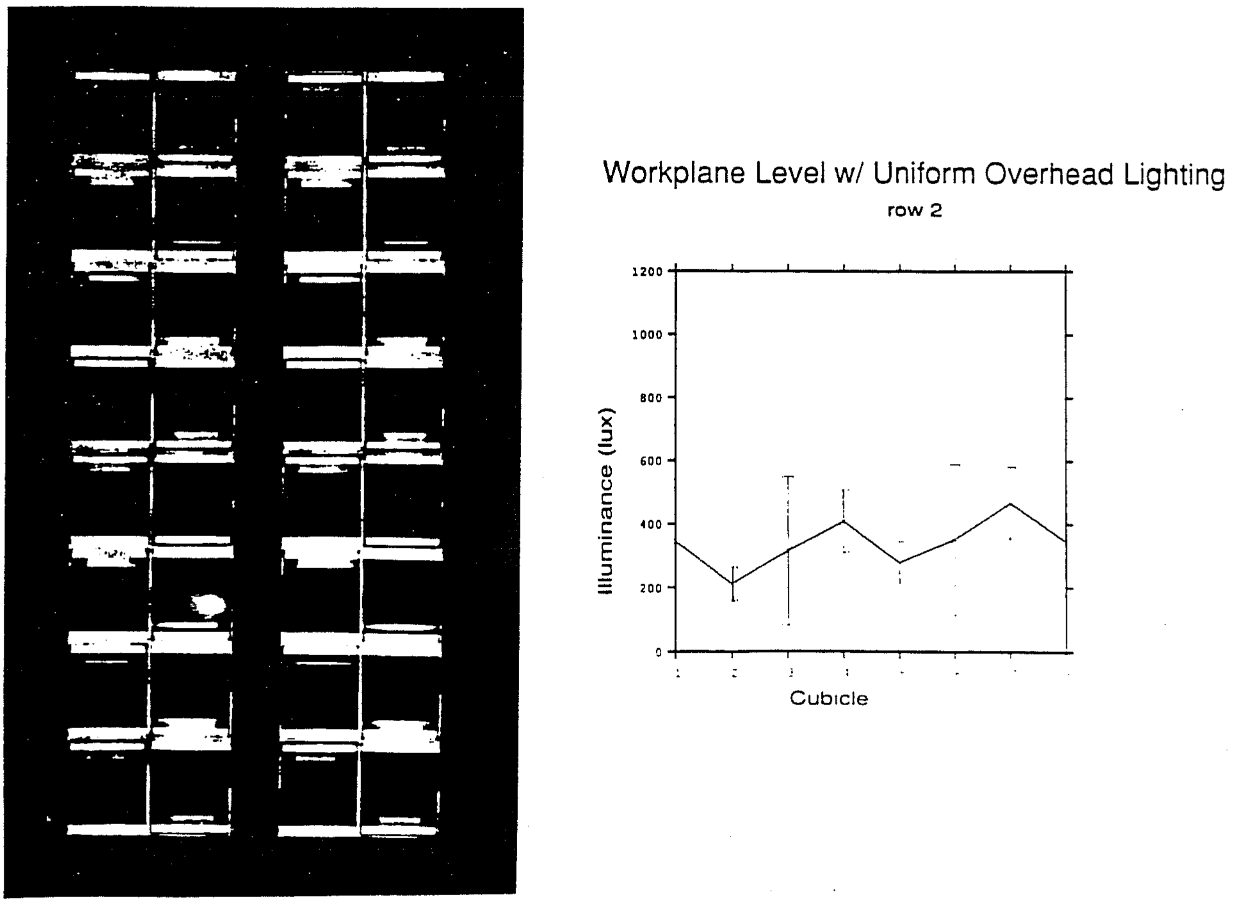


Figure 2. Partitions added to space and their effect on light levels with initial layout.

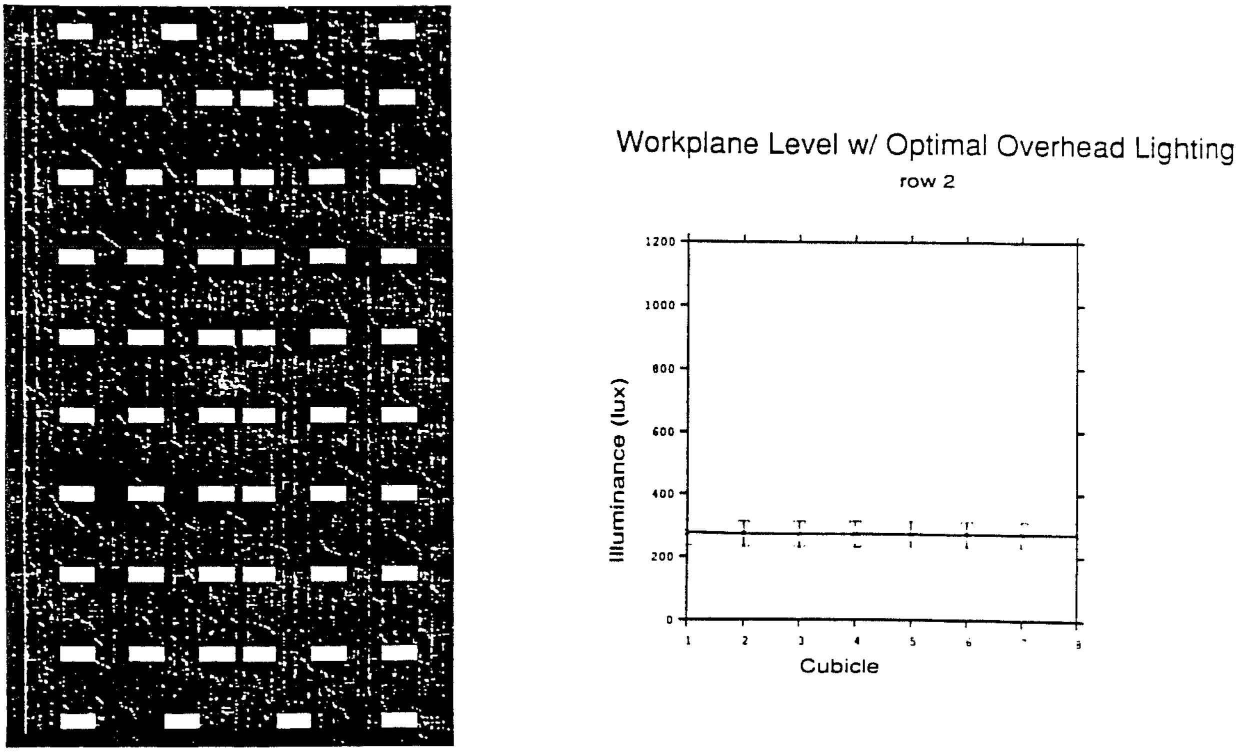


Figure 3. Optimized luminaire layout for office space with partitions.

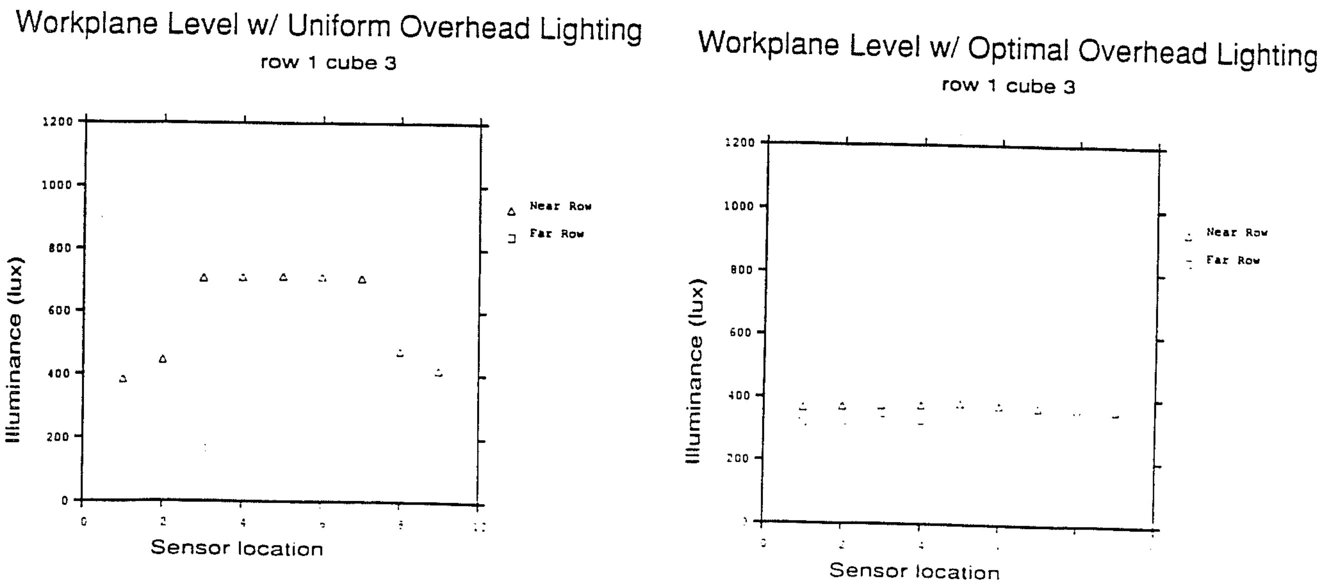


Figure 4. Desk illumination for original and optimized overhead lighting.

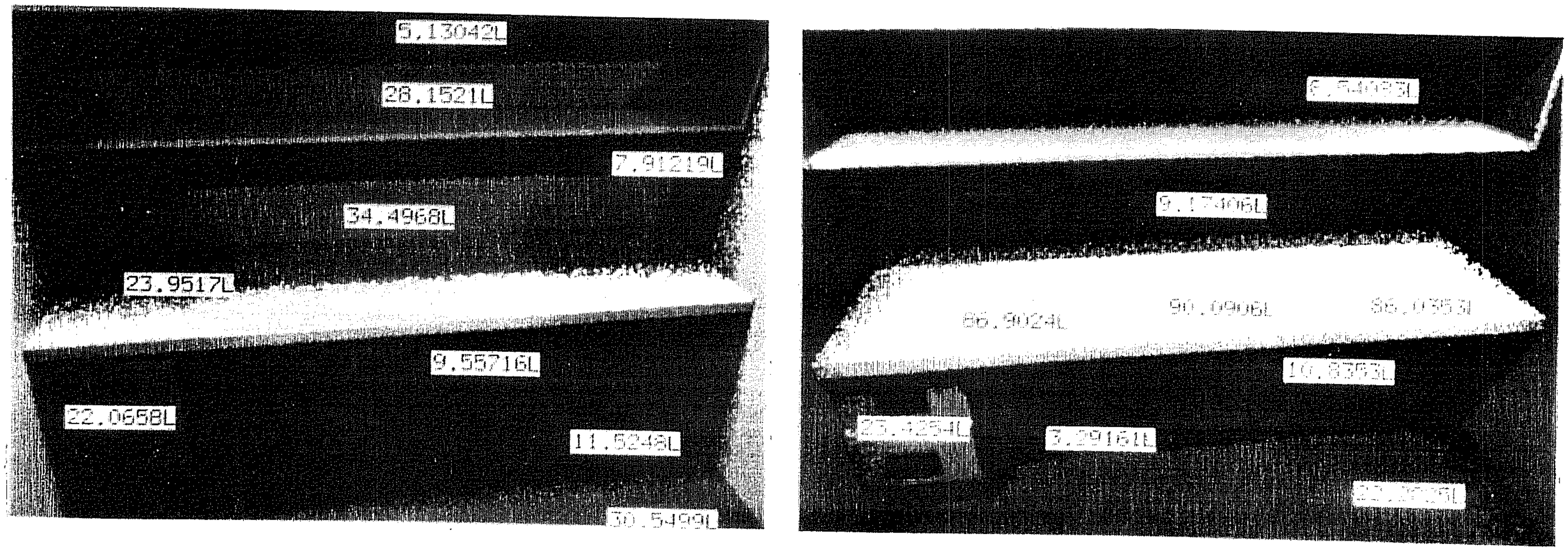


Figure 5. Vacant desk under original and optimized overhead lighting. Luminance values are in candelas/square meter.

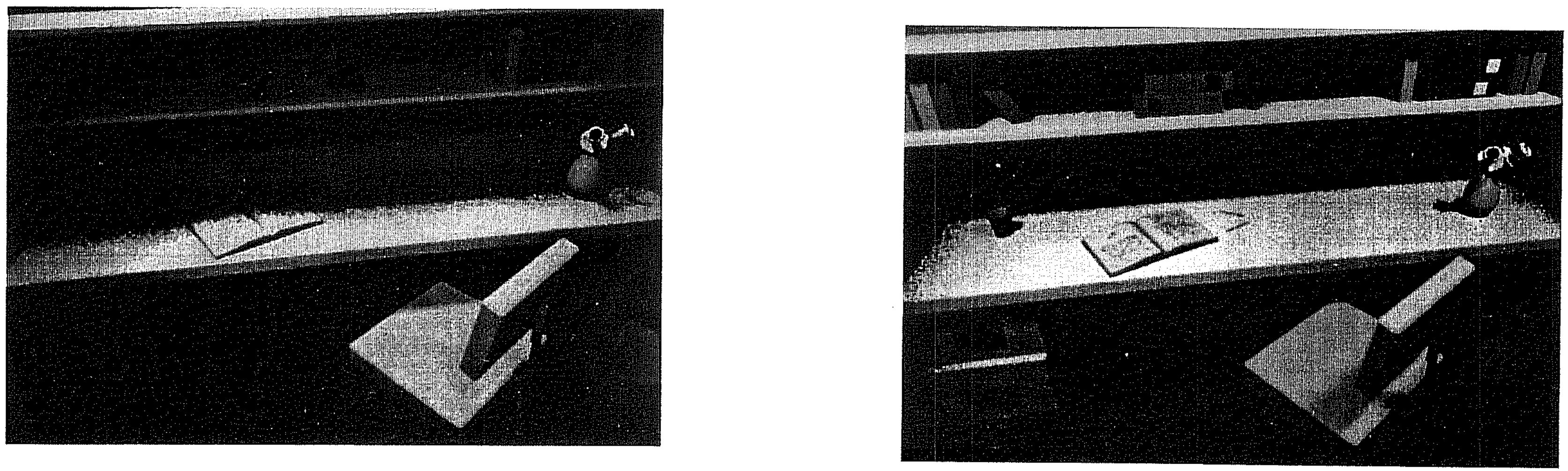


Figure 6. Furnished desk under original and optimized overhead lighting.

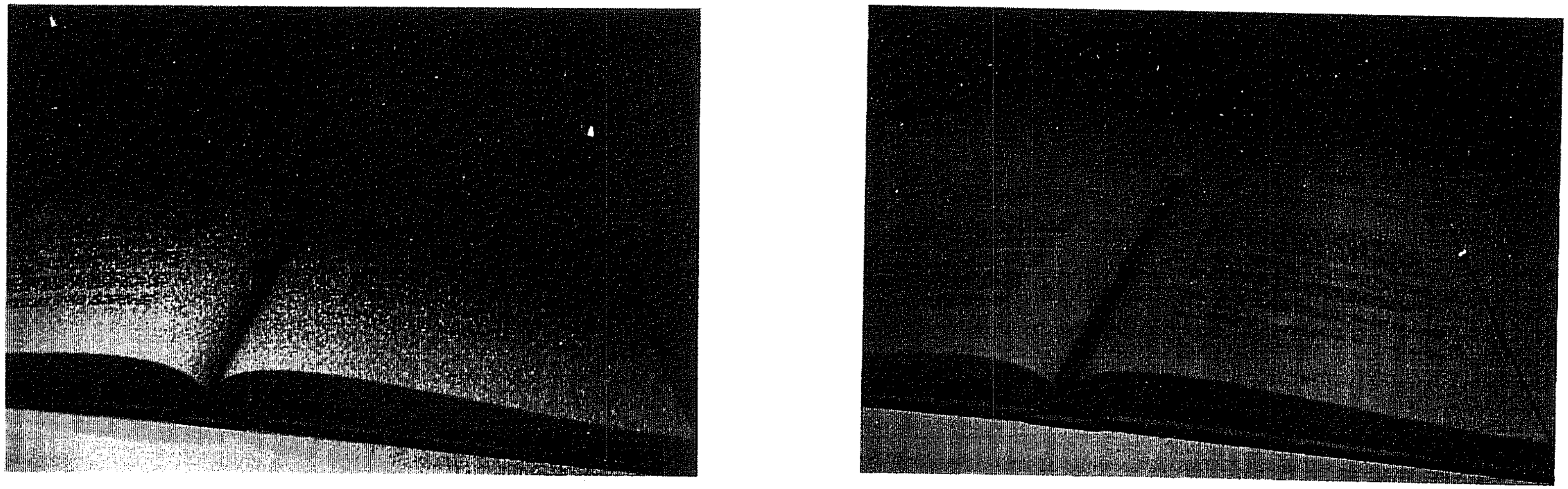


Figure 7. Reading task under original and optimized lighting

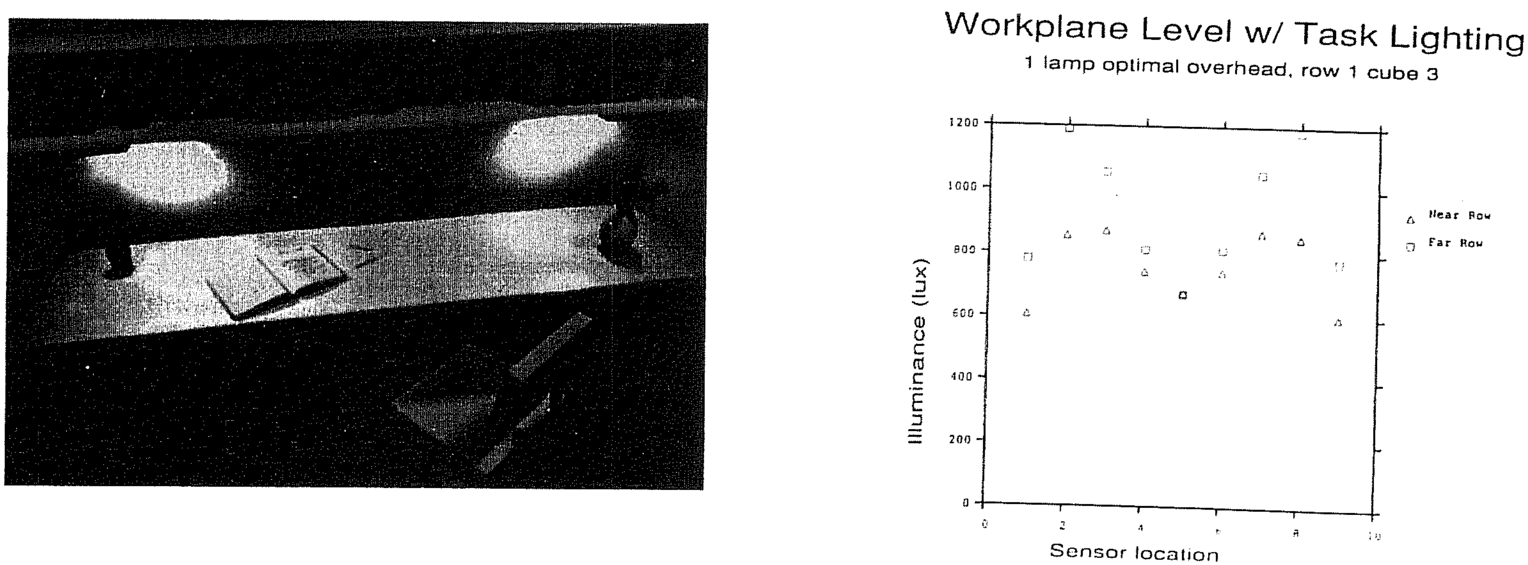


Figure 8. Desk illumination with task lighting added to optimized overhead layout.

II. Validation of the Program

1) Modeling of an Existing Place, the Conference Room

Qualitative Validation

As mentioned before, part of the validation was done last year with very good results [Grynberg88]. This part concerned the **quantitative** aspect of Radiance. However, the **qualitative** aspect, is just as important to architects, who want to predict the visual quality and atmosphere of a project. This consideration brought us to model an existing space. The *computer model* used by Radiance describes a space by its geometry and materials. We chose the conference room of our building because it has both electric light and daylight, is suitably located, and represents a reasonably sophisticated space.

Radiance

Before discussing the qualitative validation we conducted, we must first describe the software environment and how it is used. Radiance was initially written to explore new and old techniques in lighting simulation. It has some important capabilities not found in other ray tracing programs. Foremost is the ability to accurately account for both diffuse and specular interreflection in complicated spaces. This is a basic requirement for the prediction of luminance in architectural applications.

The software is available for computers (minicomputers to main frames) that run the UNIX† operating system. The software is divided into several components whose relationships are shown in Figure 8. In the center are the three main programs, *rtrace, rpict* and *rview*.

Rpict produces picture files in batch mode.

Rview previews scenes interactively.

Rtrace calculates luminance, illuminance, and other application-specific information.

The main programs are variations of the same basic ray tracer. Their input will be explained in the "Conference Room Modeling" section.

Major Commands

Xform: transforms a Radiance scene description file according to the options given. The scene can be complex (i.e. a complete description), or can concern only one object. The transformations, followed by their arguments, are executed in the order they are given.

The main options are:

-t x y z translate the scene along the vector *x y z* .

†UNIX is a trademark of AT&T corporation.

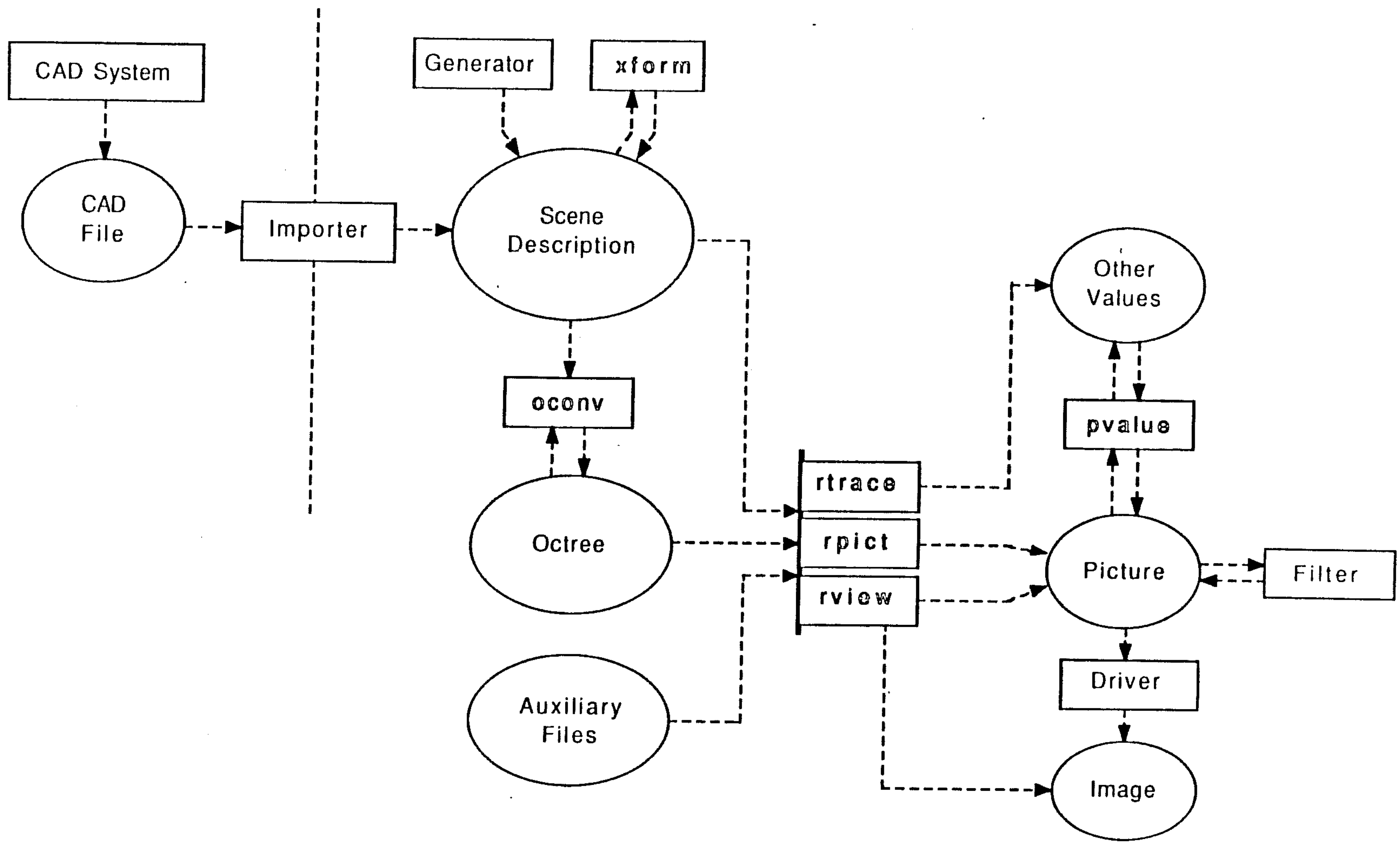

CAD System
CAD
File
Importer
Generator
xform
Scene
Description
oconv
Octree
Auxiliary
Files
rtrace
rpict
rview
Other
Values
pvalue
Picture
Filter
Driver
Image

Figure 8

-rx degrees (ry, rz) rotate the scene *degrees* about the x axis (respectively y, z axis).
-s factor scale the scene by *factor* .
-mx, my, mz mirror the scene respectively about the *yz, xz, xy* plane.

genbox: produces a Radiance scene description of a parallelepiped with one corner at the origin and the opposite corner at (x,y,z). Options are provided for beveled or rounded edges.

gensky: produces a Radiance description of the sky distribution for a CIE clear sky (with or without sun) or a CIE overcast sky (CIE stand for Commission Internationale de l'Eclairage).

genprism: produces a Radiance scene description of a prism, or extruded polygon.

pipe (|) Often, the output of a command (or program) must be considered as the input of another command (or program). The "pipe" construct ("|") is used for such "transfer" of data.

Octree and Makefile

The description of a scene is contained in an octree. Roughly, the generation of the *octree* is analogous to the compiling process for a program, where the octree file is the "executable." The octree is made by sorting the scene's information to improve efficiency during rendering. The conversion does not bring any additional information, it just arranges the surfaces in a certain way. The command to create the octree for the conference room is:

oconv conf > conf.oct

where *conf* is the input file and *conf.oct* is the octree file. The *oconv* command has several options that are described in the manual [Ward89]. The main options we used were *-i*, which allows one to add scene information to an existing octree, and saves time regenerating the octree, and the *-f* option, which produces a frozen octree.

A "frozen octree" contains all the information needed to generate a scene. When rendering with a frozen octree, the program looks only at the octree. This is why working with a frozen octree takes less time, but is also less flexible. In an ordinairy octree, the geometry described in the scene file(s) to which it refers cannot be changed. However, it is possible to change some non-geometric parameters such as the color of a material or a pattern without having to redo the octree. During rendering, the program looks at both scene description and octree, so it takes more time than with the frozen octree. For a better description and explanation of octrees see [Ward88].

During scene creation, the work completed is kept in the main file ("conf") and the work in progress in the file "new". This improves the speed for octree regeneration, since only a small part of the octree required updating after each change. When work was completed, there were several octrees and files which depended on each other, and a means for updating them automatically was desired. We therefore created a file called **Makefile**, which is read by the command "make".

"Make" is a standard UNIX command. It executes a list of shell commands associated with each target, typically to create or update a file of the same name. Makefile contains entries from targets that describe how to update them with respect to the files and/or other targets on which each depends, called dependencies. Although "make" is usually used for programs, we use it for updating octrees. It looks at the newest and the latest version of the scene files and, when necessary, regenerates each octree. In Appendix 1 is the Makefile for the conference room.

Conference Room Modeling.

The modeling of the LBL Building 90 conference room required several distinct steps. First was the description of the overall geometry and the main materials. Several surface types were used to describe the **geometry** of the scene. Basic types include *sphere, polygon, and cone.* 3381 surfaces were actually used to model the scene, although the use of instances (explained below) meant that the scene effectively contained over 13,500 surfaces. The surfaces are called "primitives" and must be declared as specified in the manual. For example, the definition of a polygon is:

```
modifier polygon identifier
0
0
3n
        x1    y1    z1
        x2    y2    z2
        ...
        xn    yn    zn
```

A polygon requires $3 \cdot n$ real values (n is the number of vertices). The order of the vertices is counter-clockwise as viewed from the front side (into the surface normal). The modifier is usually the identifier of the material the surface is made of. The basic types of **materials** include *light, plastic, metal, glass, and dielectric.* For example, the definition of the material *plastic* is:

```
modifier plastic identifier
0
0
5 red green blue specularity roughness
```

When it was easy to do so, we guessed the specularity and roughness of the materials. In other cases, like the lamps' distribution, we refered to the manufacturer's data.

a) Basic Geometry

The pictures 1, 2, 3, 4 give a view from the 4 corners of the room. The upper image is the simulation and the other is the photography of the real space.

The room is 35.5 by 22 foot rectangular, has an 8.9 foot ceiling, 4 shafts (wall extrusions) and a projection room. A plan view with dimensions of the conference room is in Appendix 5. We modeled the **walls** as *polygons* with "holes" for the doors and windows. In order to check the work, we put some test lights in the room to "see" it. Using the *rview* command, it is possible to walk around the scene and look at it from any view point, aim or reduce on precise object.

Second, we modeled the **shafts** and the **projection room,** which we considered as a bigger shaft with a window. We modeled them as boxes using the *genbox* command, which generates a box (optionally with round edges). Genbox creates a box with one corner at the origin and all positive coordinates. To move the box to the correct place, we used the *xform* command. A translation, can be given by the *-t* option and rotations about the 3 axes by *-rx and -ry and -rz* to put the object in the correct place. The center of the rotation is the origin, an important point to bear in mind when moving objects. For example, if one does a rotation followed by a translation, the origin of the object will have been moved, which can cause some confusion. A good approach is first to move the object to the origin, rotate it, and then translate it to the right place. The same remark can be made about the scale command *(xform -s)*.

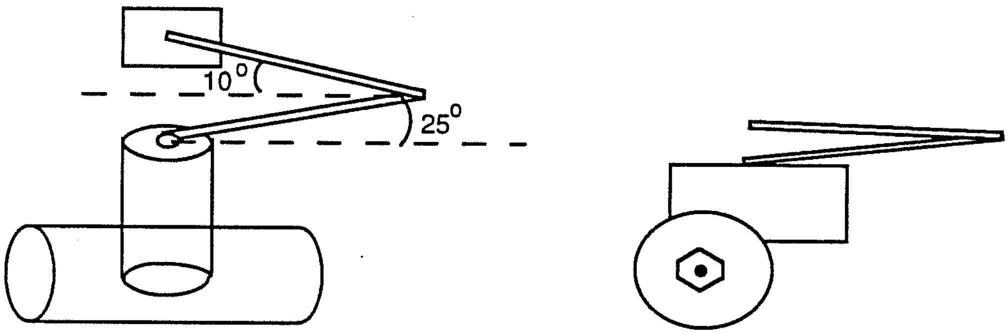
Third, we created the **baseboards,** modeled also as boxes. This may seem to be a small detail, but it makes the scene more realistic because, unconsciously, we are expecting to see them. Their presence might not be noticed, but their absence would be.

Next, we modeled the **doors** and the **door jambs.** For the door jambs, we used the *genprism* command. We gave the points in the X-Y plane and extruded the object along the Z axis, then used the xform command to place it correctly.

At this point, the order of element creation is less important because the general shape of the room is finished. One only needs to fill it with the different furniture and elements. The conference room has boards made of wood at the height of the chair backs to protect the paint on the walls. We modeled the **Wood boards** along the walls as boxes with beveled corners. The two rows of **screws** on the boards are a combination of several files. The file "screw" which describes a single screw made with a cone and two rings at each end, and the file "screws" which is a succession of xform commands placing the screws along the first row. In the main file, we used the file "screws" twice, once for the first row, and a second time for the second row. The xform command allows one to duplicate elements very easily. Because the screws are small, it is easy to "lose" them. Often, they became embedded in the board, so it took some time to get them in the proper place. But this effort greatly improves the realism of the scene.

The **door closer** was more difficult to do. It is composed of two cylinders embedded perpendicularly one in the other. A small sphere is the holding point of the arms, which are two long boxes shifted at different angles to the wall above the door (see figure 9). The door closer material is metal. Since metal is difficult to visualize with all its reflections, we made the working material into a red plastic, so it was easier to find and to look at. The door closer is contained in a separate file, like most of the elements. The file contains the description and the materials concerning it, so that all the elements can be put in a library and reused. Working this way also gives faster feedback.

<u>Door closer</u>



Front view

Side view

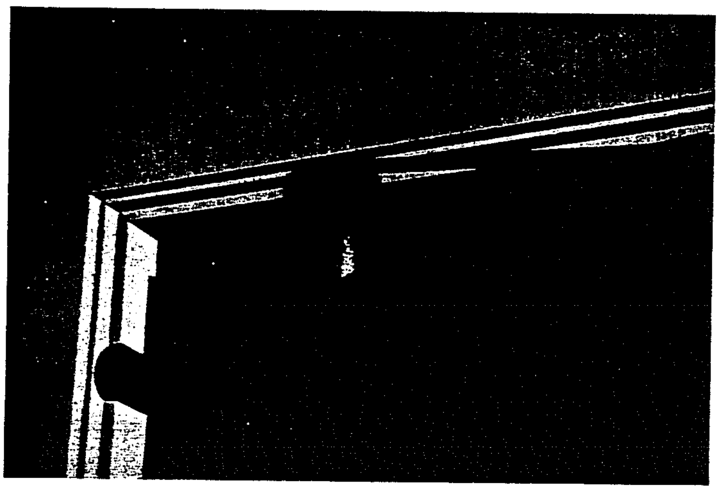


Figure 9

b) Patterns

The **blackboard** is made out of several polygons, and the **pin board** has the same geometry as the blackboard, but its material has a *pattern*. Patterns are used to modify the reflectance of materials. It is a perturbation of the material color. A procedural pattern is given as an equation that relates the pattern's value to the current intersection point, ray direction, surface normal, distance, etc. For example, the speckle pattern of the pin board uses

$$speck=if(noise3(Px,Py,Pz)\hat{}4-.5,A1,1);$$

where

- noise3 is a 3 dimensional noise function,
- A1 is the first real argument in brightfunc, the brightness of the specks,
- and Px,Py, Pz are the coordinates of the intersection point of the ray and the surface.

So if $noise3(Px,Py,Pz)\hat{}4-.5>0$ then $speck=A1$ otherwise $speck=1$.

We wanted the pin board to have "waves" so that it looked like real fabric. To have this effect we used another pattern modifying the first one:

$$bwave=1-.5\cdot(fnoise3(Px\cdot4,Py\cdot4,Pz\cdot4)/2+1)\cdot(sin(Px\cdot2\cdot PI)/3+2/3)\cdot(sin(Py\cdot2\cdot PI)/3+2/3)$$

where fnoise3 is a 3 dimensional fractal noise function. On the next page is the file that defines the pin board.

FILE	*COMMENTS*
# creation of the pin board void brightfunc pinbspeck 4 speck speck.cal -s .003 0 1 .4	"#" signals a comment. void is used when there is no modifier for the item. Brightfunc is a type of pattern. It is a procedurally defined monochromatic pattern. Note that pinbpat is modified by pinbspeck.
pinbspeck brightfunc pinbpat 6 bwave bwave.cal -s .02 -rx 90 0 0	6 is the number of arguments which can vary. bwave is the name of the reflection variable; bwave.cal the function file; the rest is the transformation: -s .02 scaling of 0.02; -rx 90 rotation along the x axis of +90 degrees.
pinbpat plastic white_mat 0 0 5 .516 .45 .34 0 .2	pinbat is the modifier of the material plastic 5 is always the number of arguments for plastic and metal. values are red green blue specularity roughness
void metal edge_mat 0 0 5 .43 .43 .43 .5 0	This metal has no modifier. The specularity is always higher than in the plastic case
white_mat polygon pinboard 0 0 12 0 .05 0 6 .05 0 6 .05 4.04 0 .05 4.04	Geometric definition of the pinboard as a polygon.

```
!genbox edge_mat board_right .062 .052 4.04 | xform -t 5.938 0 0
!genbox edge_mat board_left .062 .052 4.04
!genbox edge_mat board_top 6 .052 .062 | xform -t 0 0 3.978
!genbox edge_mat board_down 6 .052 .062
```

Comments: Because it is a command, genbox must be preceded by an exclamation sign to be interpreted correctly. The general form of the command is

genbox material_name object_name Xsize Ysize Zsize [options].

The result is a parallelopiped's description with one corner at the origin and the opposite corner at (Xsize, Ysize, Zsize). Here, the description is "piped" into xform in order to move it. The output of genbox is transfered to xform as its input. Although in this case only a simple translation was necessary, several transformations can be done in succession. It is advisable to decompose a transformation rather than doing "everything at once," because it is easier to understand and to correct.

c) Surfaces defined by functions

To generate the **curtain,** we used *gensurf*. Gensurf can be used to describe any kind of functional surface. The 3 coordinates are functions of 2 variables, s and t. In the case of the curtains, the y function is a sine function where x and z are linear functions. We have:

$$!gensurf\ void\ curtain\ 's \cdot 12'\ '.3 \cdot sin(24 \cdot 2 \cdot \Pi \cdot s)'\ 't \cdot 6'\ 400\ 1$$

The material's type is void because its definition lies somewhere else. S and t vary from 0 to 1 in steps of 1/400 and 1, respectively. The surface is composed of $400 \cdot 1$ quadrilaterals, which are rectangles in this case. Along the z-axis, the surface is unbroken because it does not vary in this direction, and there is no need to subdivide.

The command above gives the description of only a third of the entire curtain. We divided the curtain into three sections, because the conference room actually has three curtains. We wanted to be able to leave them "open" or "closed" to generate pictures and comparisons with varying amounts of daylight. Another reason is that we can use instances to save data storage on two of the three curtains. An *instance* is a compound surface, given by the contents of an octree file. Reusing an octree in this way can save a lot of space.

The modeling of the **shaft guards** is interesting because each shaft guard is made out of one "sophisticated" polygon. The shaft guard is a long rectangle with one rounded corner. The rounded part is interesting, and we approximated it as a juxtaposition of regularly spaced vertices. To create a set of points regularly spaced, the *rcalc* program is very efficient. We used it to create the shaft guard polygon. Rcalc calculates mathematical expressions from files *(-f option)* or on command line *(-e option)*. It reads from an input which can be anything. For regular points, the input is best given by *cnt*. Cnt is a simple program which generates a list of N numbers from 0 to N-1 when *cnt N* is given. By scaling these numbers, it is easy to obtain regularly spaced points. The shaft guard has a rectangular part and a rounded part which is exactly a quarter circle (figure 10). Rcalc was used for the creation of this quarter circle.

Shaft Guard

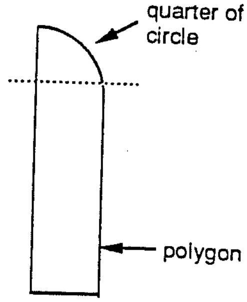


Figure 10

The command line was:

```
cnt 10 | rcalc -e 't=($1+.5)/10*PI/2' -e '$1=.208*cos(t) ; $2=0' \
        -e'$3=2.29+.208*sin(t)'
```

Cnt 10 will produce 10 numbers from 0 to 9 that rcalc will read and t is a variable which represent the angle. As was mentioned earlier, the shaft guard is made out of a single polygon. The upper right point of the rectangular part is already defined, so we exclude it in the definition of the rounded part. Thus why we have t=($1+.5) instead of t=$1.
We want t to vary from 0 to $\pi/2$, so we scaled t by 10, which brings the range from 0 to 1, and then by $\pi/2$, which brings the range to 0 to $\pi/2$.
Rcalc will create three columns ($1, $2, $3) which will be the three coordinates of the points. If "*cnt M N*" is given, the input will be two sets of numbers:

M	N
0	0
0	1
..	..
0	N-1
1	0
1	1
..	..
M-1	0
M-1	1
..	..
M-1	N-1

This allows rcalc to take its input from two variables.

d) Furniture

The most difficult objects to model were the chairs whose shape is difficult to describe in terms of equations. The following explains how we modeled the **black chair** (picture 5).

The origin is set at the lower left extremity of the front left leg. The axes are such that most of the chair has positive coordinate values. The frame is done with several "boxes". The back frame has an "H" shape that is shifted by a certain angle. As explained before, rotating shifted objects is a difficult job because determining the exact coordinates and keeping track of the origin is very difficult. This is why it is highly recommended to decompose the shape into simple elements and manipulate these elements as clearly as possible. In our case, we could have shifted, one by one, every "box" of the H, however this would mean require computing every coordinate of every piece, after it was shifted. That introduces calculations of $\sin\theta$ and $\cos\theta$ which may have many digits. The number of digits entered affects the results, and entry errors can cause many calculation difficulties. It is much easier to shift the H all at once.

The arm rests are made with genbox using the -r option. They are modeled as wood boxes with round edges, and the wood is a pattern. The back frame and the front frame are connected with a rounded piece. To describe this rounded piece, we used *genprism* because it allows us to describe a surface in the X,Y plane and extrude it in the Z plane. Genprism can either read points from the standard input or from a file. We approximated the round piece to the part of a circle, measured the radius and, by having the starting and ending points of the piece, we could figure out the angle of it. Then, by using *rcalc*, we obtained the points we needed to generate the surface. Once we established the surface in the X,Y plane, we had to rotate it to have it in the X,Z plane and finally to translate it so that it the back and front pieces of the frame are connected.

The material used for the seat and the back of the chair is "plastic." The seat is approximated to a flat box, but the back is rounded. Again, to model the back, genprism was used. Here, two rotations, one along the Z axis and one along the X axis, were needed.

e) Lighting

There are three kinds of lights in the conference room :

- daylight
- luminaires
- dim light of the exit signs.

To simulate **daylight,** it is necessary to model a (CIE standard) sky condition and select a time and a day of the year for the simulation. (so that a sky distribution is affected). The Radiance's primitive to create a sky is *gensky*.

The light distribution from the **luminaires** was given by the manufacturer. A copy of it is in the appendix as well as the Radiance data file. The description of the fixture needs a primitive of the program called *brightdata*. Brightdata uses an interpolated data map to modify a material's color. The map is n-dimensional. Brightdata refers to the fixture data file. The output of brightdata is the fixture distribution. This input is tedious, but not difficult. Then the identifier of brightdata is affected to the modifier of *light* which is the basic material for self-luminous surfaces. It is defined as a Red Green Blue radiance value (watts/rad2/m2). The luminaire itself is a simple polygon. In Appendix 2 is a Radiance example of the *fixture file* and in Appendix 4 a copy of the technical documentation of a luminaire.

The last kind of light used in this scene is the dim light of the **exit signs.** Because the illumination from these signs is so low, they are modeled using the type *glow*. Glow is generally used for surfaces that are self-luminous, but limited in their effect. In addition to the radiance value, a maximum radius for shadow testing is given. In our case the maximum radius was zero, since we did not want the signs to cast shadows.

A font was used to simplify the modeling of the sign lettering. The frame and the main box for the exit sign were modeled with genbox.

f) Room Measurements

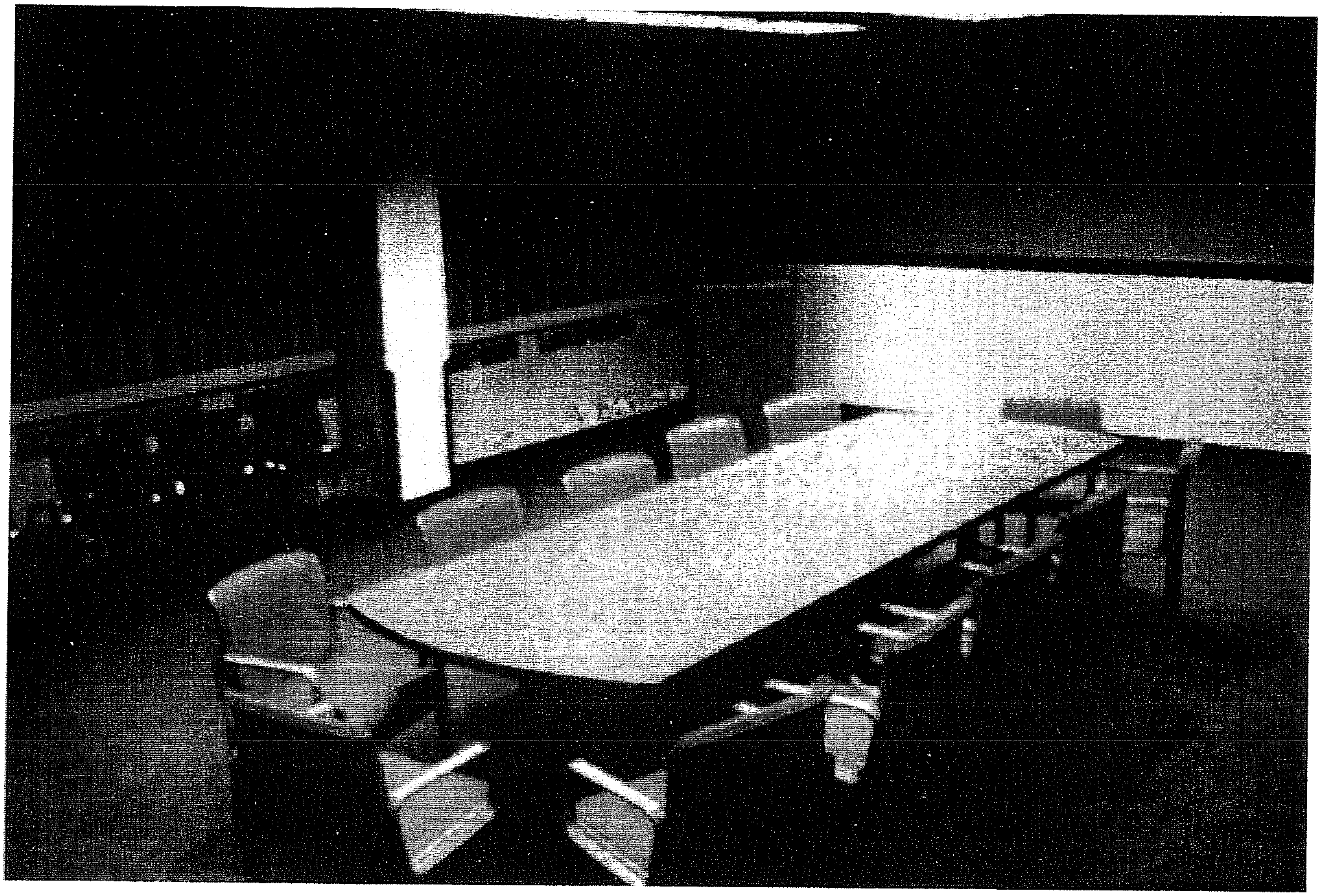
The following table shows the chosen reflectance and brightness values for the main materials and objects:

Surface	*Reflectance*
Walls	60%
Floor	5%
Ceiling	98%
Table	8%
Red Chair	7%
Green Board	25%
Pin Board	45%
Door Wood	36%
Big Shaft Wood	44%

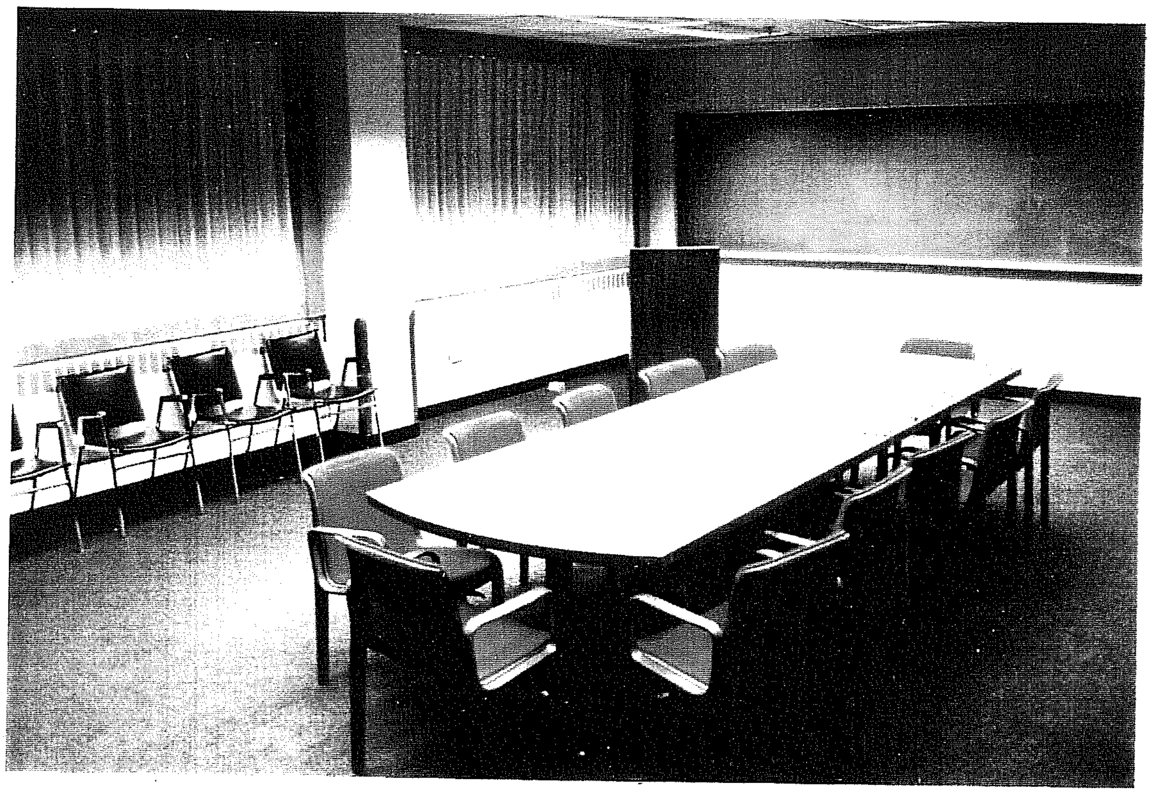
Object	*Brightness*
Red Exit Sign	74 cd/m2
Luminaires (down)	1450 cd/m2
Window (Sky)	4600 cd/m2

Conference Room

Simulation



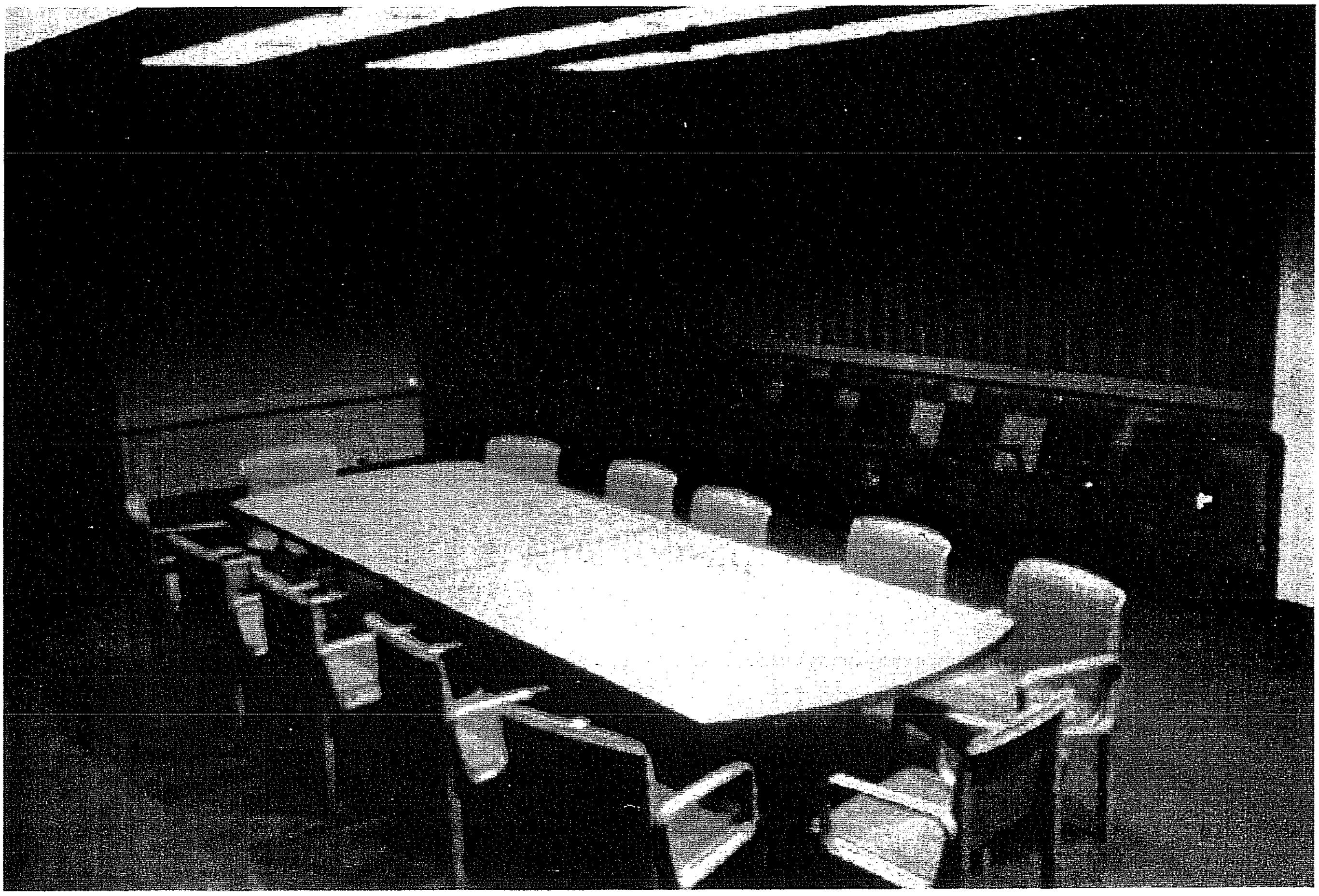
Actual



Picture 1

Conference Room

Simulation

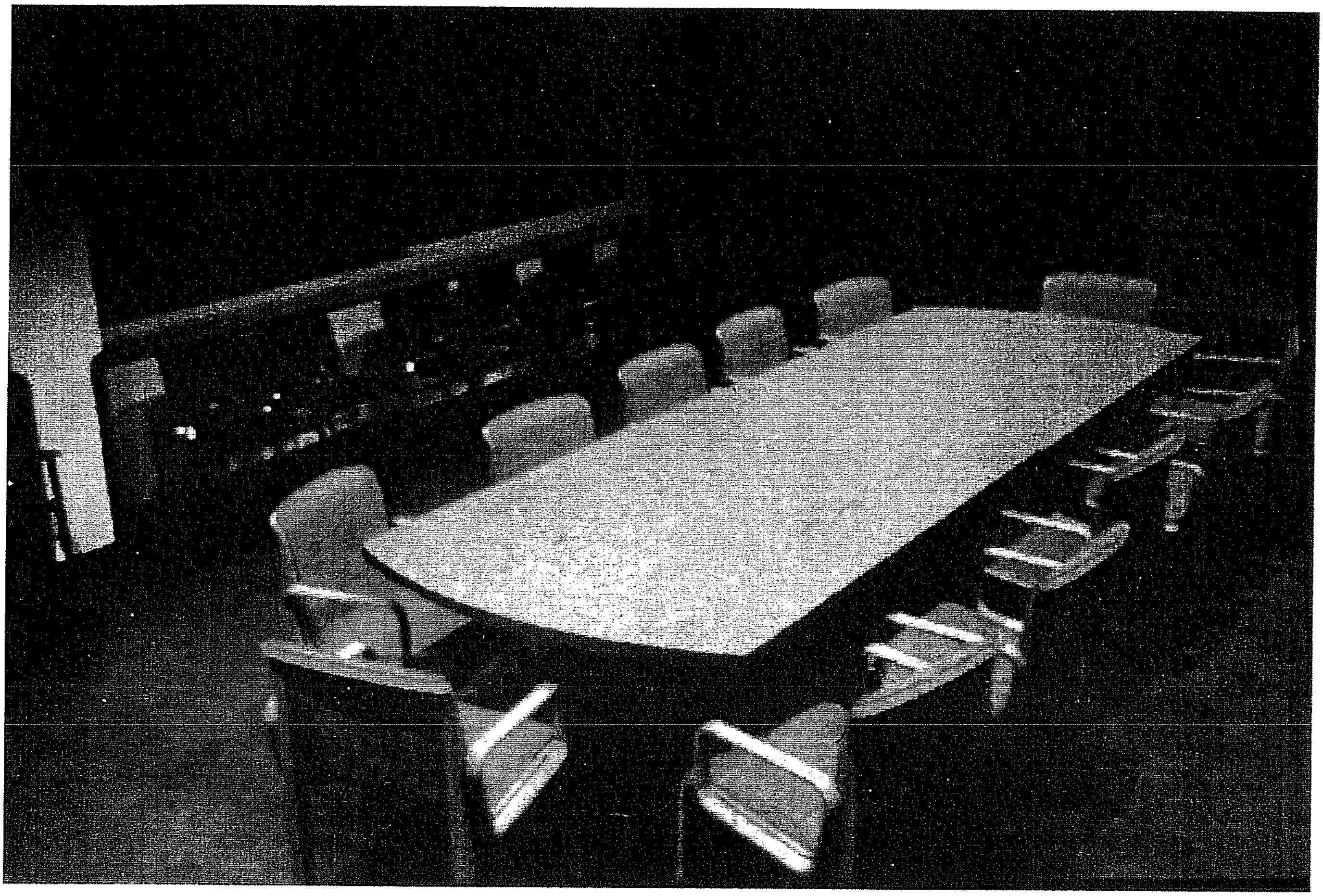


Actual

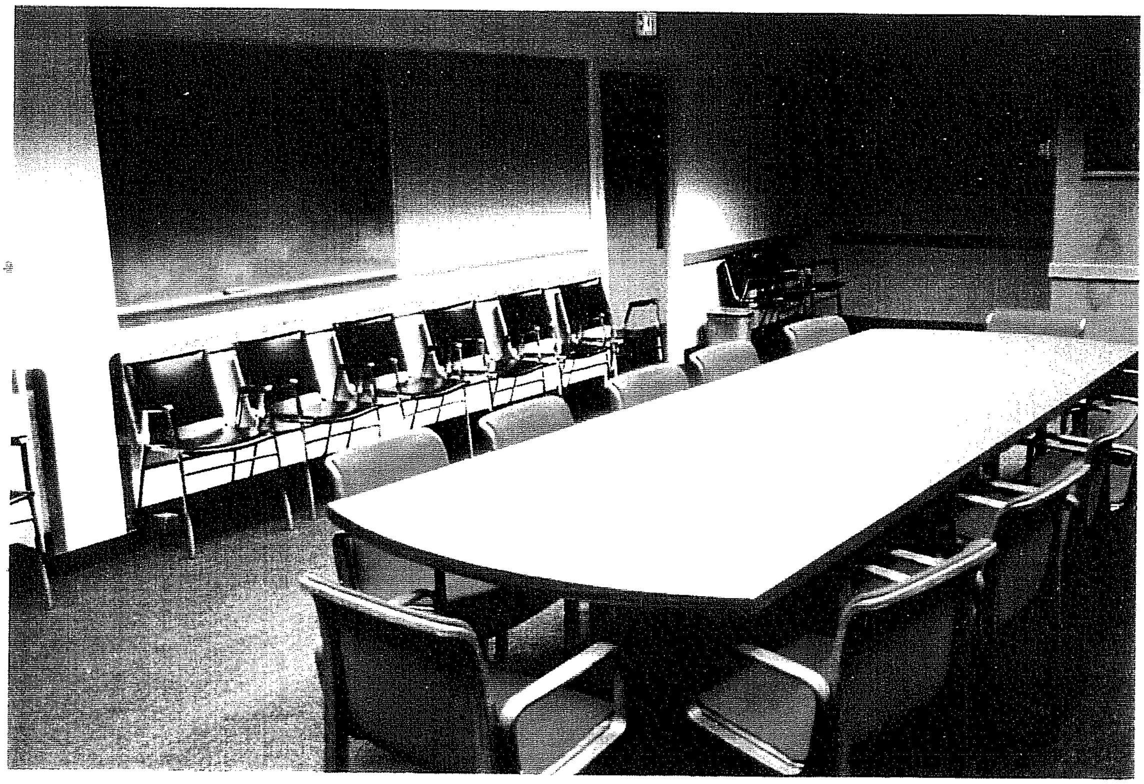
Picture 2

Conference Room

Simulation

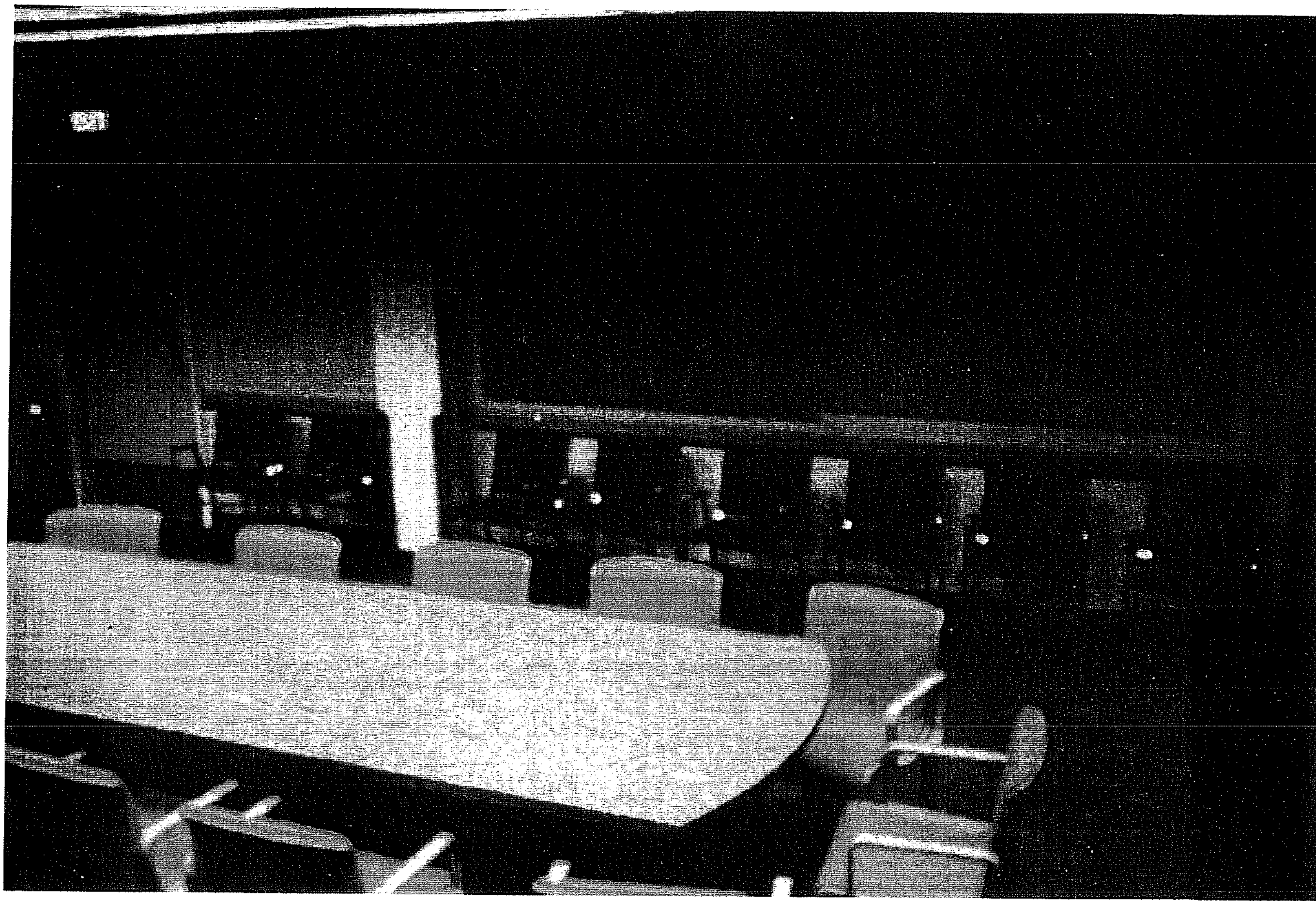


Actual

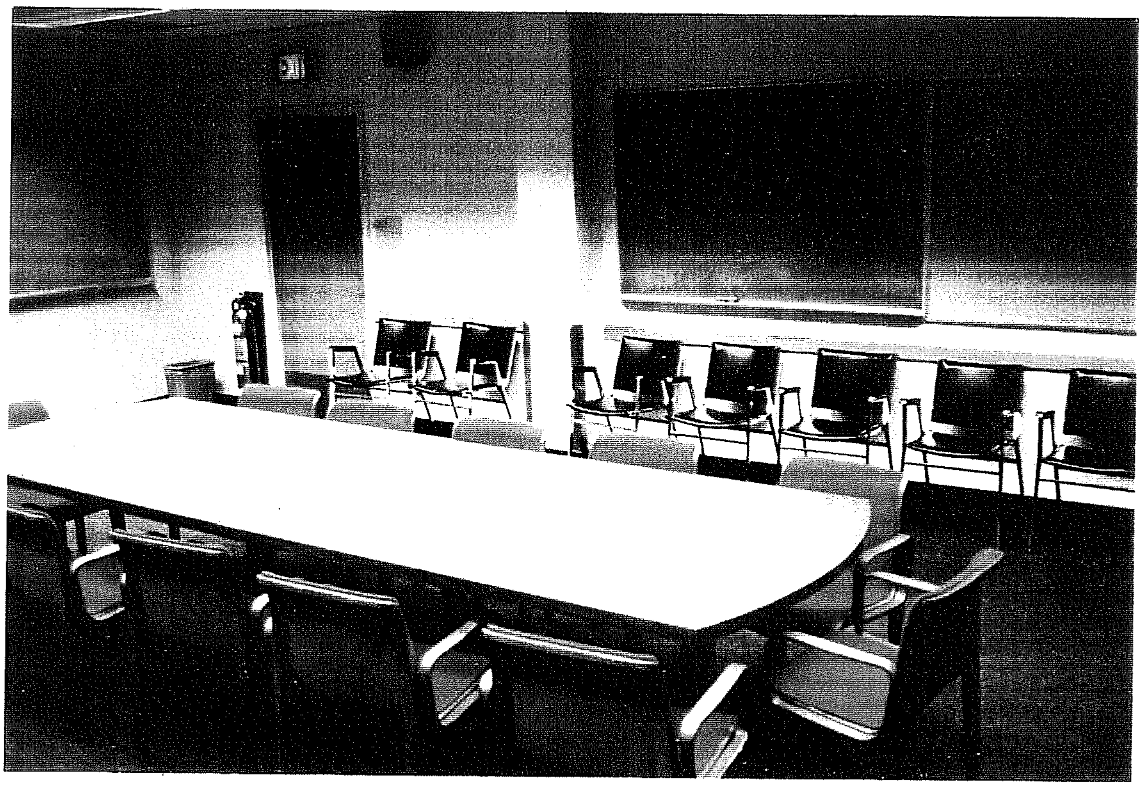


Picture 3

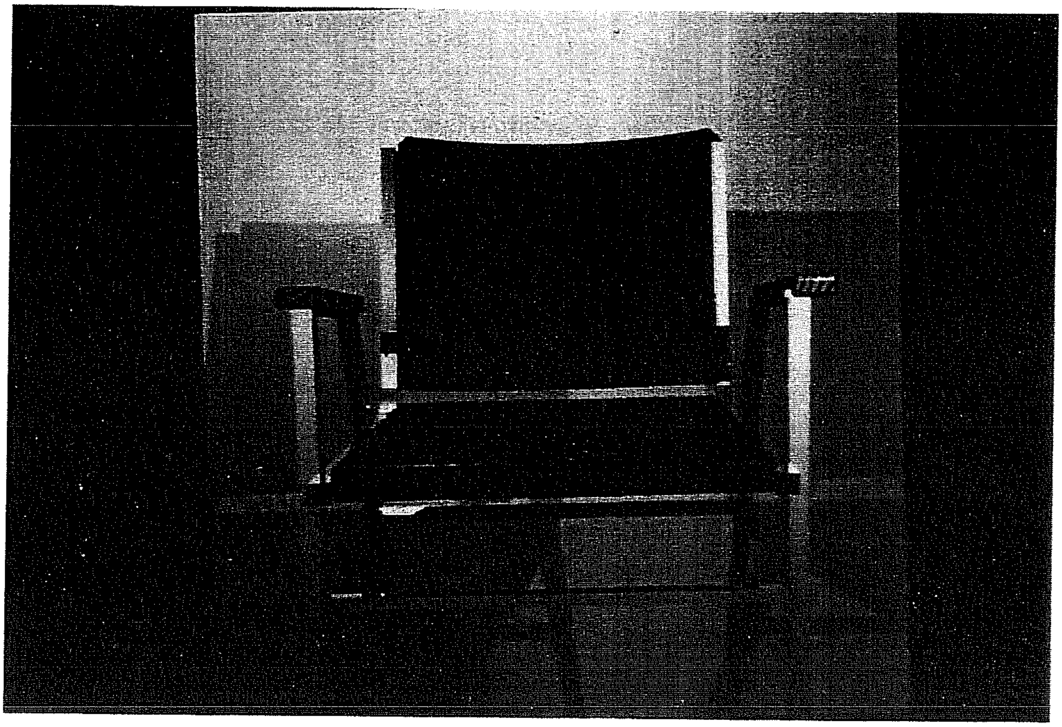
Simulation



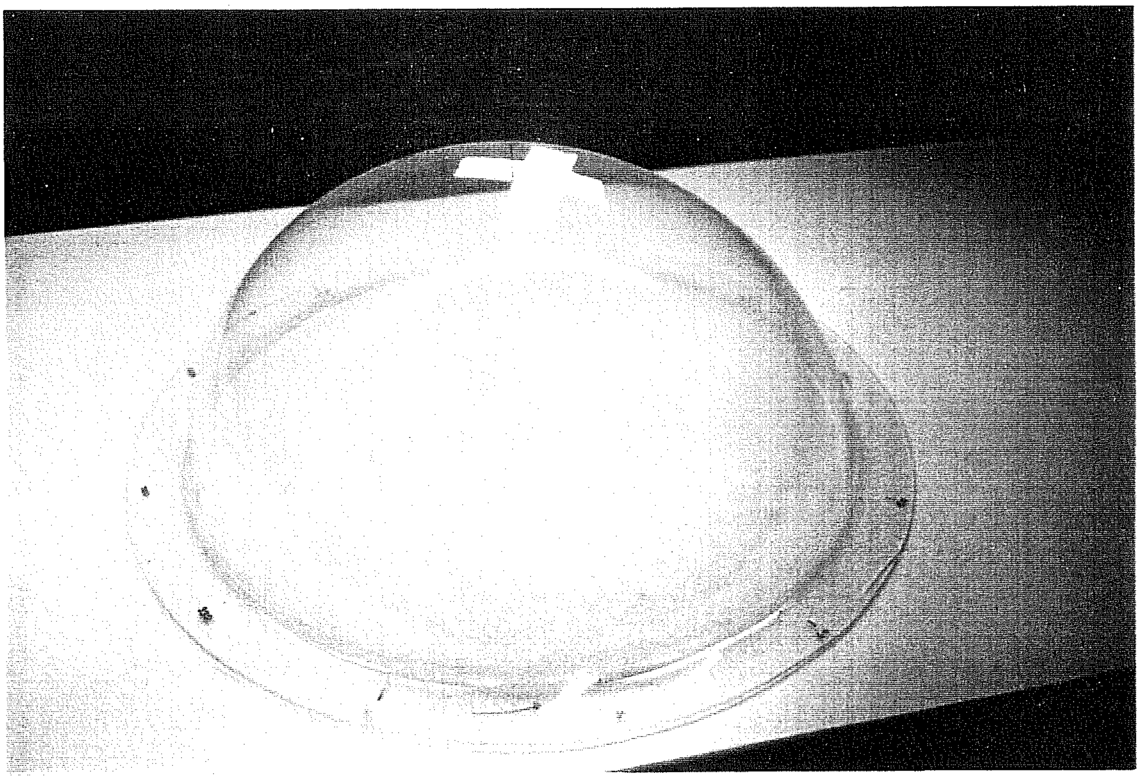
Actual



Picture 4



Picture 5: The black chair.



Picture 6: The prototype reflectometer.

2) Reflectometer

Need for a Special Tool

Exact specularity of materials is rarely known. Manufacturers rarely give or even know this characteristic. Nevertheless, these characteristics are taken into account in the Radiance's ray tracing calculations, and for that we must give approximate numbers. For a big scene like the conference room, having an exact number might not be too important. But for closer views, it could change the overall appearance.

There is no tool yet which allows us to measure the specularity of materials quickly and accurately. That is why we wanted to build the reflectometer, which is based on a simple idea.

Using a CCD camera with a fish-eye lens, it is possible to capture instantly an entire hemisphere of incoming radiation. If the fish-eye lens is coupled to a mirrored hemisphere, it is possible to capture the hemisphere of outgoing radiation from an illuminated sample (figure 11). The conversion from the image to a map of outgoing radiation, and then to the reflectance function of a material for one input source angle, is a simple computation which can be calibrated to the sensitivity of the CCD element and to the distortion of the lens/mirror system.

But, realizing that such a device is expensive, and having no certainty that it will work, we decided to build a simple plastic prototype. Plastic is convenient because it is very specular and is easy to shape.

We had the plastic hemisphere (fig 12-picture 6) done at a workshop, but the hemisphere was not perfectly spherical. We took measurements of this prototype to see if we could use it anyway. The main concern was: can we work with such a distorted sphere, or do we need a more perfect one? Also, if some distortion is acceptable, how much of it can we allow?

Analytical Representation of the Sphere.

To answer to these questions, we tried to get a numerical representation of the prototype sphere. Since our program Radiance can handle spheres well, we thought of doing a Radiance representation of the distorted sphere as a perfect sphere with a perturbation of the surface normal. We measured the radius for several orientations, fixing a reference as the center of measurement. Phi and Theta were chosen as shown in Figure 12. The measurements we got are in Appendix 6. Figure 13 shows the radius measurements. The general shape of the curve is parabolic instead of linear, partly because the sphere is distorted, but mostly because the measurements were not taken from the real center. We used these data to find the best center and radius of the experimental sphere with a program we wrote for that purpose, *center.c*, which uses the fitting method of χ^2 [Bevington]. (The methods of least squares and multiple regression are restricted to fitting functions which are linear in the coefficients.)

We can define a measure of goodness of fit of a function $y(x)$, χ^2:

$$\chi^2(x_i) = \sum\left\{\frac{1}{\sigma_i^2}[y_i - y(x_i)]^2\right\} \tag{1}$$

Where the σ_i are the uncertainties in the data points y_i.

The method of least squares consists of determining the values of the parameters a_j of the function $y(x)$ which yield a minimum for the function χ^2 given in equation (1).

Grid Search

If the variation of χ^2 with each parameter a_j is fairly independent of how well optimized the other parameters are, then the optimum values can be determined most simply by minimizing χ^2 with respect to each parameter separately. This is the method of the *grid search.* With successive iterations of locating the local minimum for each parameter in turn, the absolute minimum may be located with any desired precision. Along each parameter's axis, we approximate the χ^2 function to be a parabola.

The general formula for a parabola through three points is:

$$P(a) = \left[\frac{\left[\frac{P_2-P_1}{a_2-a_1} - \frac{P_1-P_0}{a_1-a_0}\right]}{a_2-a_0}\cdot(a-a_1) + \frac{P_1-P_0}{a_1-a_0}\right]\cdot(a-a_0) + P_0 \tag{2}$$

where (a_0,P_0), (a_1,P_1), (a_2,P_2) are the 3 points, and $P(a)$ is the equation of the polynomial.

The center of the spere will be the minimum of the parabola for each parameter.

For a parabola, the extremum will be found when the derivative of the above equation is equal to 0.

The solution of $\frac{dP(a)}{da} = 0$ is :

$$a_{ext} = \frac{1}{2}\left[-\frac{P_1-P_0}{a_1-a_0}\cdot\frac{a_2-a_0}{\left[\frac{P_2-P_1}{a_2-a_1} - \frac{P_1-P_0}{a_1-a_0}\right]} + (a_1+a_0)\right] \tag{3}$$

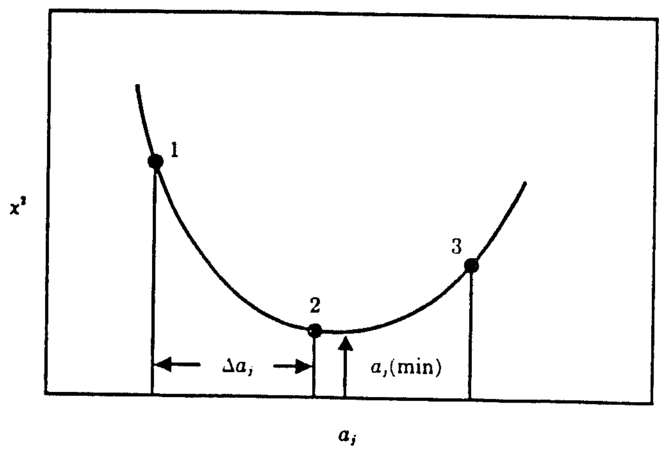
The procedure of the grid search we followed is:

1) Start with 3 points $a_0 = a_i - \Delta a$, $a_1 = a_i$, $a_2 = a_i + \Delta a$, where a_i is the initial point and Δa is the initial spacing.
2) Compute χ^2 for each of the 3 points.
3) Approximating χ^2 as a parabola, compute the second derivative as follows:

$$\frac{d^2P(a)}{da^2} = 2\cdot\frac{\frac{(P_2-P_1)}{(a_2-a_1)}-\frac{(P_1-P_0)}{(a_1-a_0)}}{(a_2-a_0)} \qquad (4)$$

If the second derivative is positive, indicating that the extremum is a minimum, then use the value given by equation (3) as a'. Otherwise, use a value for a' smaller than a_1 if $P_1 < P_3$ or greater than a_3 if $P_3 < P_1$.
4) Insert a' into the set a_1, a_2, a_3 such that a' is always used, the largest previous value is discarded, and $a_1 < a_2 < a_3$.
5) Repeat this procedure for each parameter in turn.
6) The whole procedure above is repeated until the difference between a' and a_2 is less than the acceptable error, ϵ.

The following program uses this method of grid-search to find the "best" center (X,Y,Z coordinates) and the radius (R) of the distorted sphere.



Parabolic interpolation to find position a_j(min) for minimum χ^2 using the values of χ^2 for $a_j(1)$, $a_j(2)$, and $a_j(3)$.

Center.c

The program itself is in Appendix 7. Here is the description of center.c where we use the grid search method explained above.

Input: file with the numerical data taken experimentally on the "sphere".

Output: location of the "best center" of the sphere as well as the radius.

For the sphere, χ^2 is:

$$\chi^2 = \sum_i |(x_i - X)^2 + (y_i - Y)^2 + (z_i - Z)^2 - R^2|$$

Where x_i, y_i, z_i are the points measured on the sphere; R the radius; X, Y, Z the center of the sphere.

We follow precisely the grid search method explained above.

The center of the sphere will be the minimum of the parabola for each parameter.

Because we don't have a real parabola, we need to do this procedure several times until it converges to the best point. Once we have found a min_point we keep 3 points out of the 4 (the 3 we used to find the parabola plus the min_point). We chose the 3 closest to the center point. We do this procedure for each parameter until it converges correctly.

Experimental sphere model

Once radius and center are given, it is possible to make a model of the experimental sphere to see were the reflections will take place. In order to give to Radiance the model as a sphere with a perturbed surface normal, the vector normal at equally distanced points has to be found. To make this calculation, we wrote a program called *normal.c* which is in Appendix 8. In *italics* are the names of the functions to which this explanation refers.

Basically, normal.c reads the data points which are given always for the same angles of Theta and Phi. *Acquisition* takes three points and finds the circle they are generating using *circumcircle*. The middle point is the one for which the vector normal is calculated. The equation of the line the two extreme points are generating is calculated, as well as the line perpendicular to it. The direction of the perpendicular line is the direction of the vector normal to the middle point. This vector normal is calculated in the X - Z plane, and in the X - Y plane. The combination of the two vector normals for a given point creates a three dimensional vector normal. Two functions, *vnorm_theta* and *vnorm_phi*, choose the three correct points according to the plane were the calculation is made and normalize the final vector. Special cases are considered for the beginning and ending points by *first_point* and *last_point*.

Radiance Model

A comparison of the theoretical normal vector and the calculated one was done. The difference is the perturbation of the normal. These results were used in Radiance to generate a model. Several plots (figs 14, 15, 16, 17) were done to study the feasibility of the reflectometer.

Figures 14 and 15 show respectively for the distorted and for the perfect sphere the simulated image of what a sensor would see when the distribution from a diffuse surface reached it after passing through a camera with a fish eye lens. By "diffuse surface" we mean a surface that emits equally in all directions. We can see that there is a big difference between the perfect and distorted spheres, and that we don't get much of the image with the distorted sphere. The reason is that the distortion causes light to "miss" the sensor.

If a ray hits the diffuse surface with a certain angle, it should reach the sensor with the same angle. The curve of the emitted angles versus the received angle should be a line. Figures 16 and 17 show the angles emitted versus the angles received respectively for Theta and Phi. In both cases, the curve is not a line and, moreover, there are several possible emitted angles for one received angle.

Analysis of the results.

The analysis of figures 14, 15, 16, 17 shows clearly that the principle of such a reflectometer is valid, but doesn't allow much distortion. If we want the reflectometer to work, the surface has to be the closest to a perfectly spherical and smooth sphere. We hope to find a shop that will build us such a sphere, and to produce this highly promising reflectometer.

Imaging Gonioreflectometer

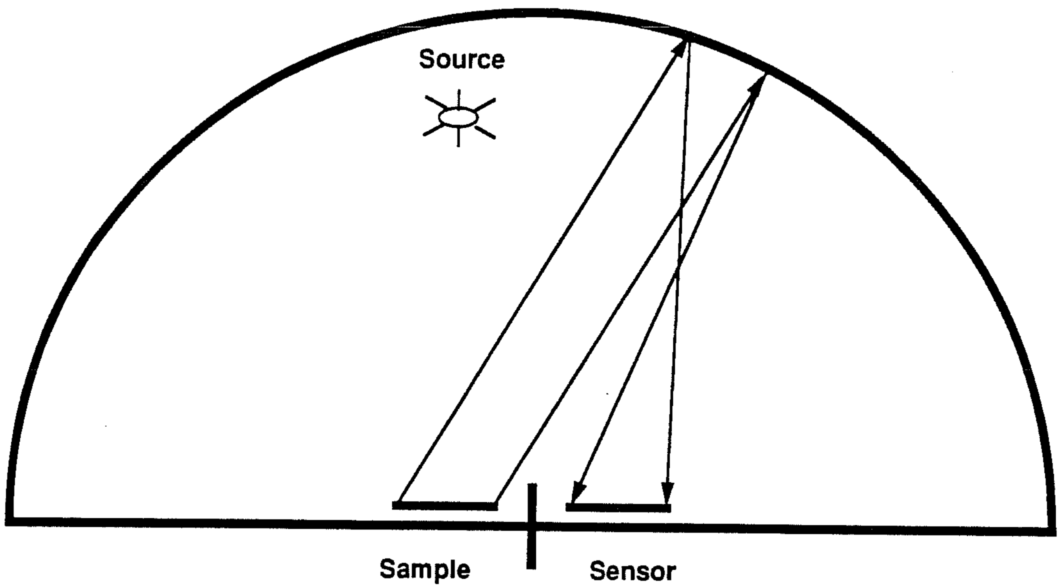


Figure 11

Schematic Representation Of The Sphere

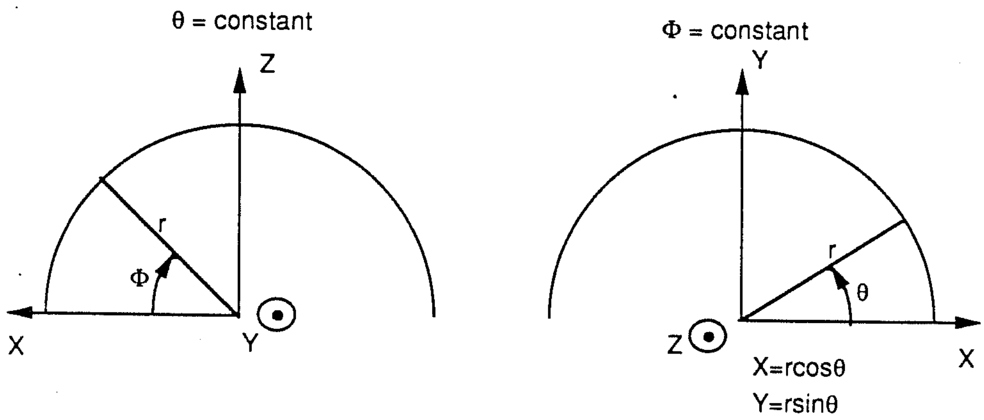


Figure 12

Radius of the distorted sphere

Radius is plot for each fixed Phi, versus Theta.

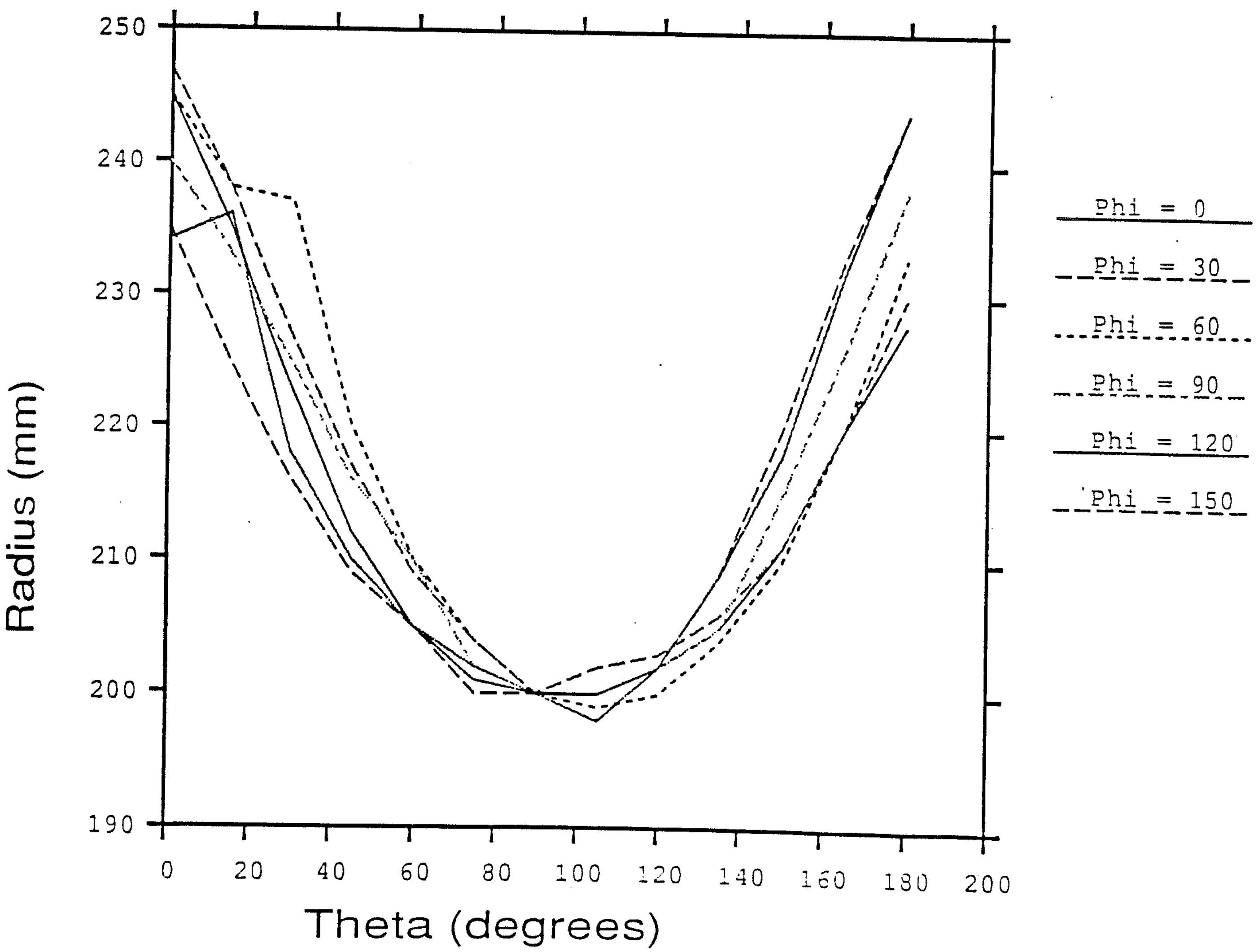


Figure 13

Distribution from Distorted Sphere

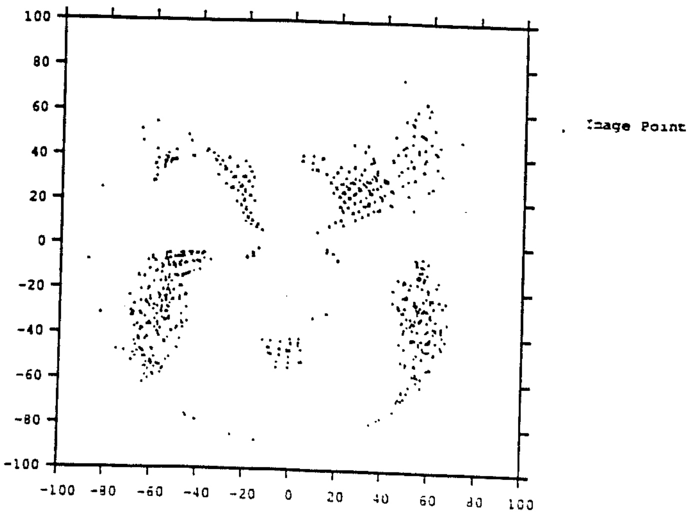


Figure 8

Distribution from Perfect Sphere

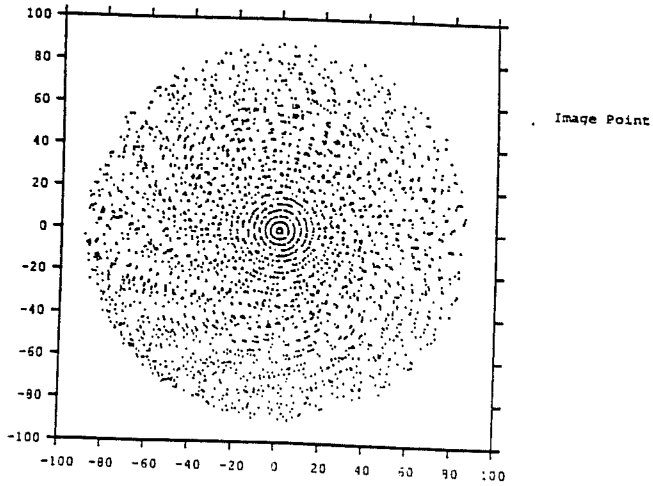


Figure 9

Emitted vs. Received Angles

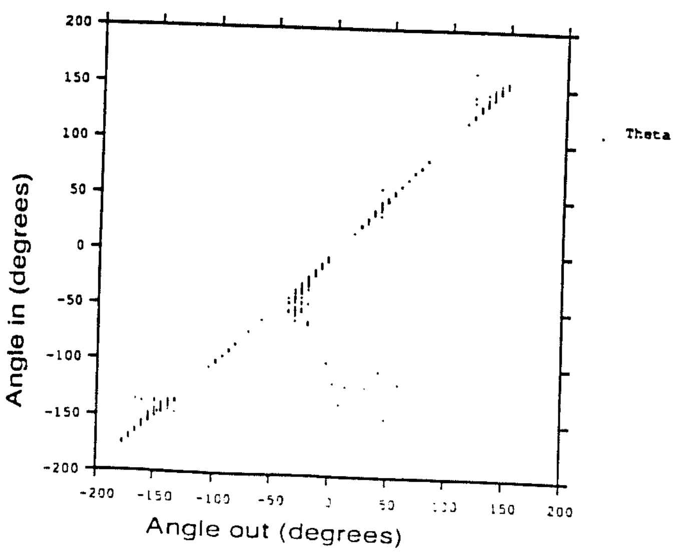


Figure 10

Emitted vs. Received Angles

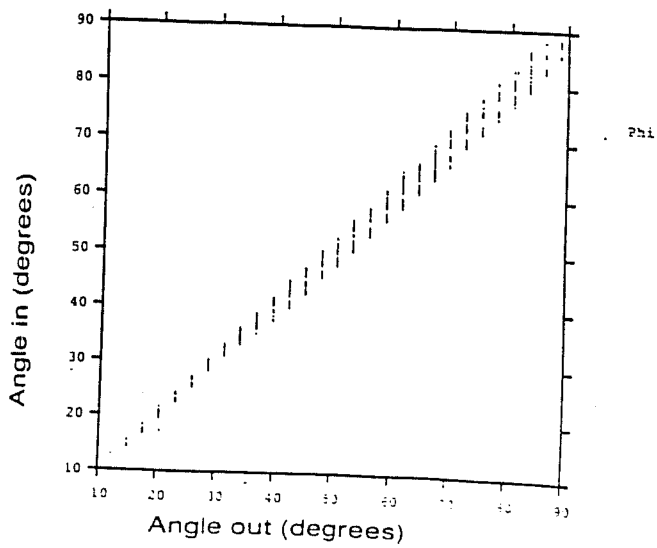


Figure 11

III. Radiance Applied to a Real Project: Israël Museum.

Light and Art

Great artists have known how to use the interplay of light, shadow, and color to enhance the aesthetic effect of their work. What they have not always been aware of is the adverse effect that light, especially ultraviolet radiation, could have on paintings and other art objects. The lighting designer, Eliyahu Ne'eman, a visiting researcher with Applied Science Division's Windows and Daylighting Group, has special interest in Museum Lighting, which he says involves "an exciting combination of physics, physiology, technology, and art".
At the question "*Why are we interested in museums ?*"
He answers:

> Some people interested in a subject feel others should join.
>
> One of the most successful ways to attract the public is to exhibit interesting objects in suitable locations - Museums and Galleries.
>
> Museums and art galleries compete for the attention, time and money of the public against other means of entertainment, leisure and commercial "exhibition" of objects in department stores.
>
> In most cases visitors pay admission fees, and demand good value for their money.
>
> Utilizing visual and, to a lesser extent acoustic media, museum designers must create displays attractive, understandable and interesting enough not only to the aficionados, but to the general public.

These are the main reasons why we should make Museums and Galleries attractive places: to increase the number of visitors and to have them enjoy the exhibits. But, in order to enjoy an exhibit, the visitors must be able to concentrate on it and therefore, they must be able to see it. Also, a comfortable and pleasant atmosphere should be provided. As a lighting consultant, Professor Ne'eman emphasizes daylight and eye comfort but also targets preservation.

Indeed, while lighting is very important for the visitor, it can deteriorate objects. Some of them cannot be exposed to regular light but only to extremely dim light. For example, the 2000 year old Dead Sea Scrolls which were found not long ago began to deteriorate when exposed to normal daylight. Professor Ne'eman designed the lighting for their exhibit using a very dim source. Most paintings cannot be exposed to direct sunlight and moreover cannot be exposed over a certain radiant exposure without becoming damaged by optical radiation.

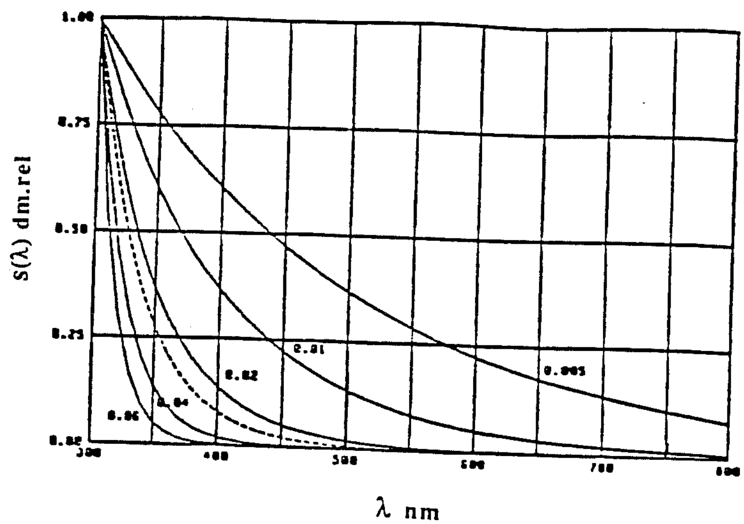
The effective irradiance E_{dm} is the amount of light reaching an object convolved by the spectral responsivity of the object:

$$E_{dm} = \int_0^\infty E_{e_\lambda} \cdot s(\lambda)dm \cdot rel \cdot d\lambda \quad ,$$

where E_{e_λ} is the spectral irradiance, and $s(\lambda)dm{\cdot}rel$ the relative spectral responsivity of the object.

The following figure (fig 18) shows the relative spectral responsivity $(s(\lambda)dm{\cdot}rel)$ as a function of wavelength (λ_{nm}) for various materials.

Relative spectral responsivity s(λ) dm.rel as a function of wavelength λnm for various materials



The damage by optical radiation is measured with the effective radiant exposure H_{dm}:

$$H_{dm}=\int_t E_{dm}\cdot dt$$

Where E_{dm} is a constant. We end up with:

$$H_{dm}=E_{dm}\cdot t$$

This clearly shows that optical damage is directly due to too much light reaching the object and that the effect increases linearly with time.

Professor Ne'eman is currently contributing to the extension of Jerusalem's modern art museum, Israël . He has designed a special skylight that lets as much daylight enter as possible, but no direct light. Direct light is dangerous in museums because it is a source of very high light levels that can damage artwork, as discussed above. That's why only a certain amount of light (direct or indirect) is authorized in a museum.

Consequently it would be sensible for lighting designers or architects to predict light levels *before* construction begins in order to make design adjustments as necessary.

Modeling with Radiance

Here our contribution takes place. A modeling of the upper floor of the museum was done using the Radiance program, so that the designers could *see* what it would look like when it was finished, and also have *numerical data* concerning the luminance on the most important surfaces. The pictures X Z Q W are runs of Radiance on the Museum file where the surfaces have the following reflectances:

Surface	*Reflectance*
Inner Wall	70%
Ceiling	70%
Floor	30%

Surface	*Transmittance*
Glass	88%

To analyze the results, we used a program called *ImageTool* developed by NCSA [NCSA] which runs under *suntools*. Among other capabilities, ImageTool can plot contour curves. It draws a curve to equal pixel values. Because pixels have luminance value in a Radiance image, curves of equal pixel values are curves of equal luminance value. The figures 19, 20, 21, 22 show the equal luminance plots for the 4 views.

According to the IES (Illuminating Engineering Society [Kaufman81]) recommendations, the light level in a museum should be 60 lux. With an average luminance of 400 lux, the numerical results show that the luminance level is over 6 times too high!

This is a very important issue that will be taken into account by the designers because of the damage that too much light causes to art objects, as we said before. The designers must make changes in order to reduce this effect.

An important parameter which influences this luminance level is the material of the floor. In order to decrease the overall luminance, the floor has to be less specular and darker.

We intend to make some runs with different values for specularity of the floor and for the walls, different sky conditions and for several days of the year.

long.fin.500x338

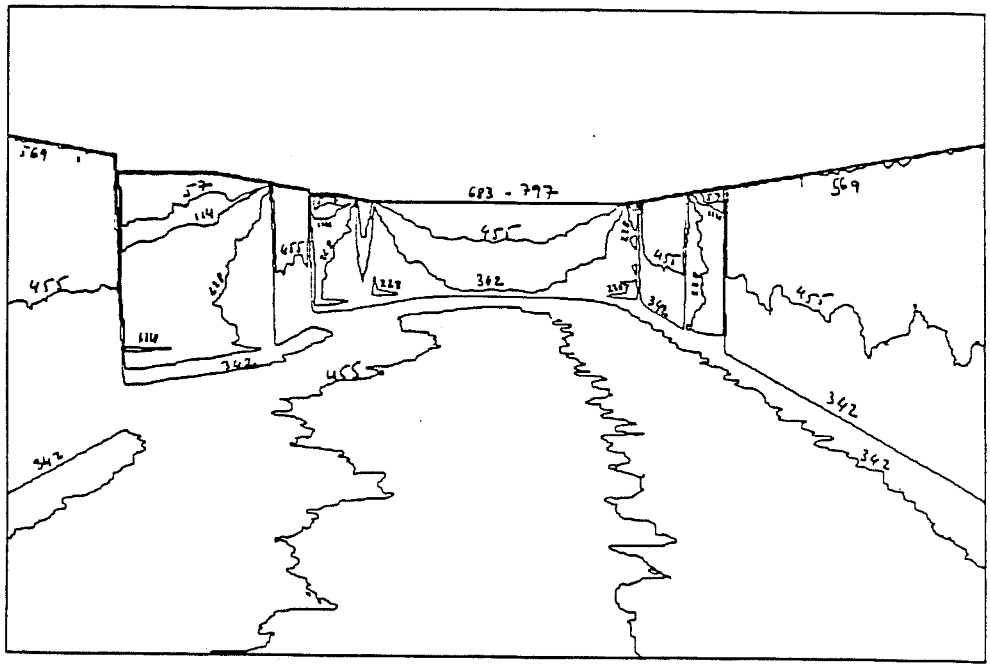


Figure 19.

lower.fin.500x338

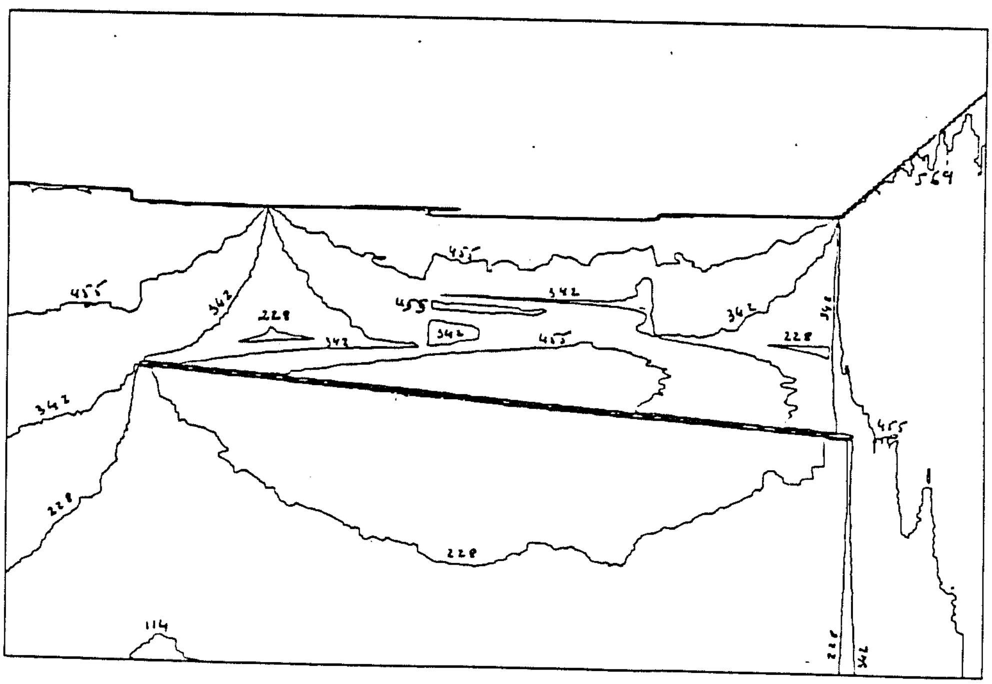


Figure 20

values are luminances in candela/m2

long_vert.fin.500x338

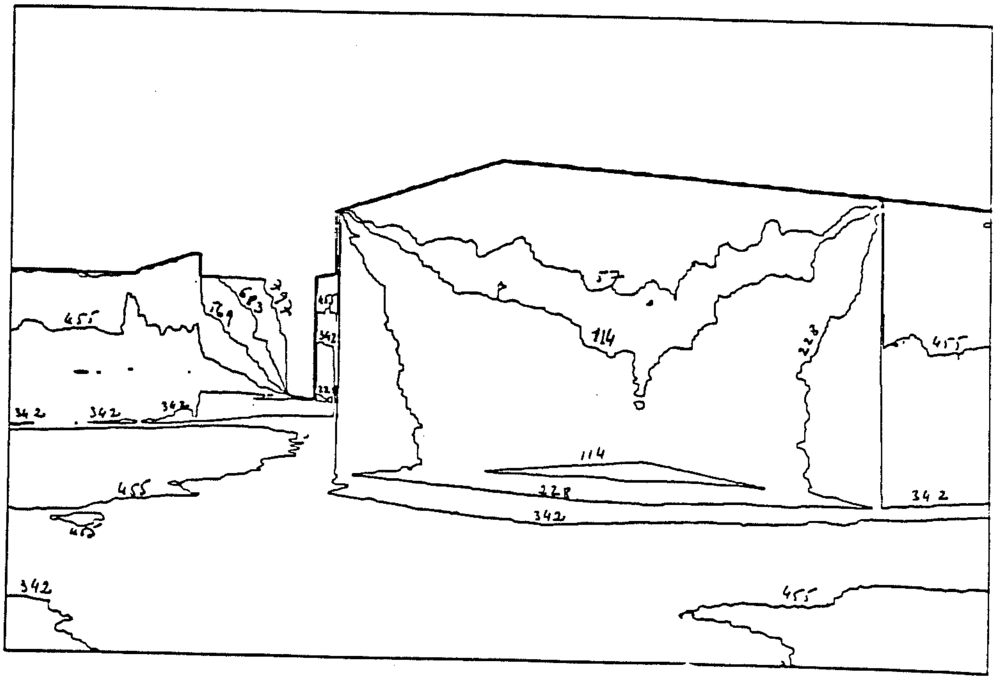


Figure 21

aLcove2n3.fin.500x338

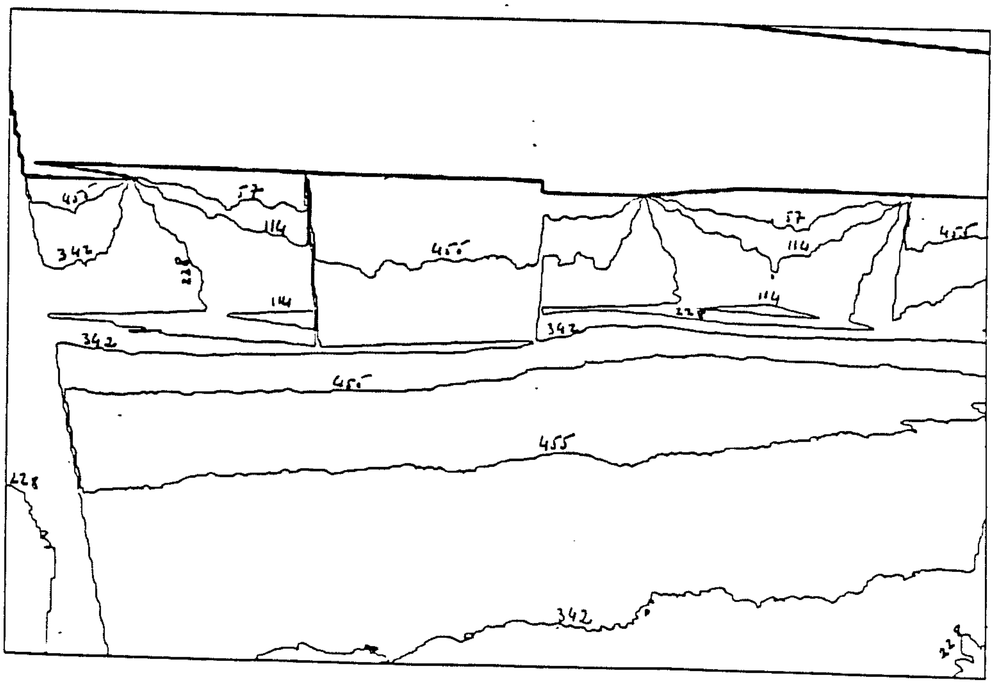
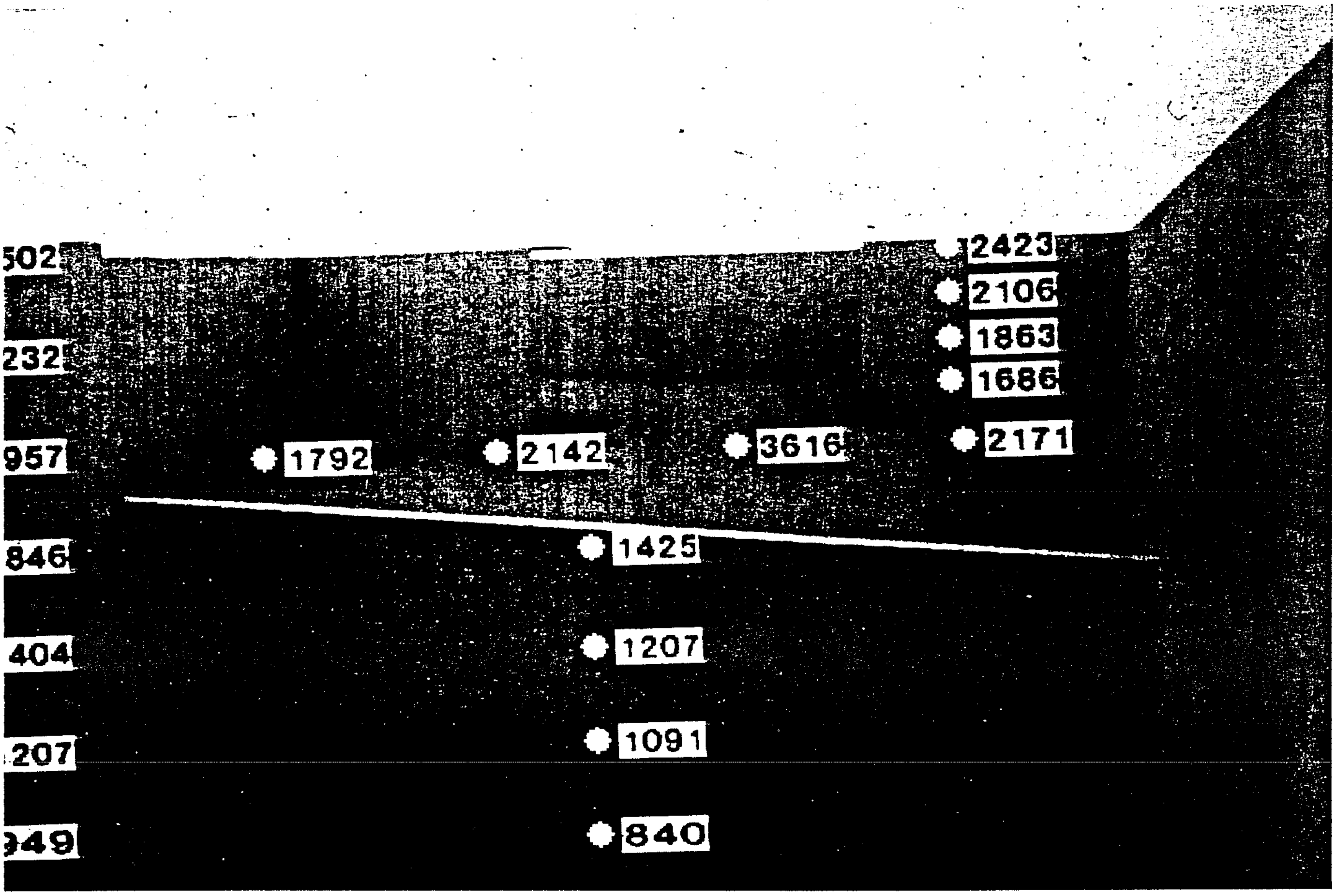
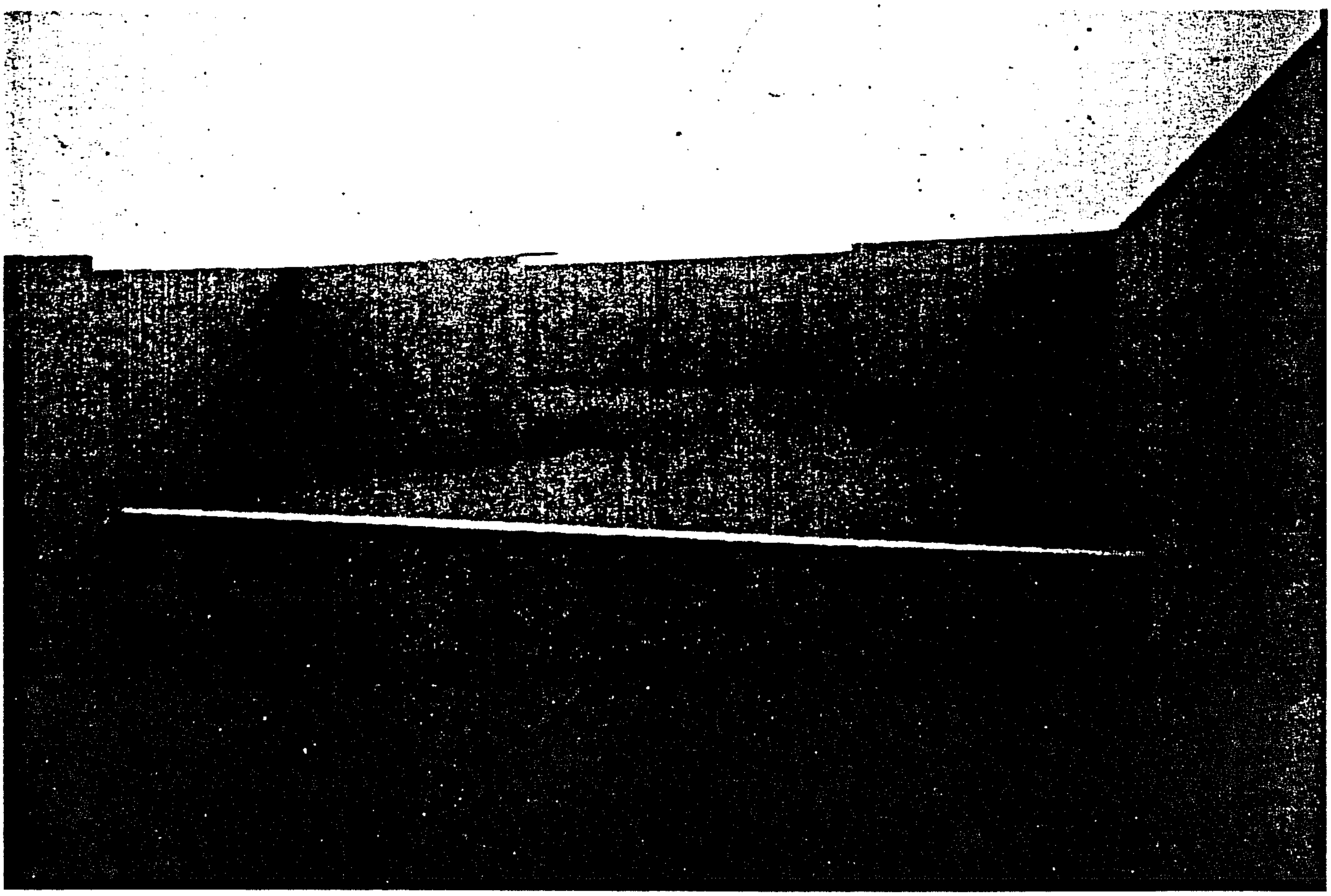


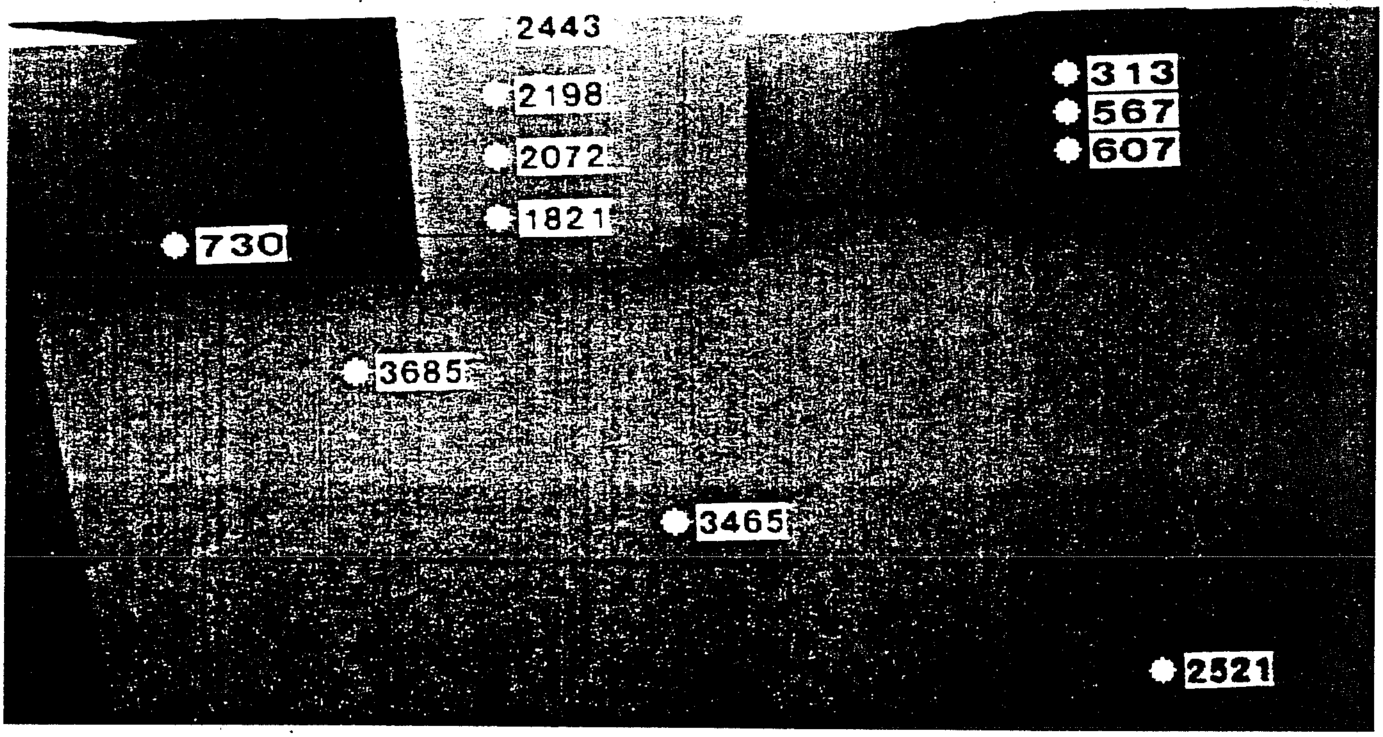
Figure 22

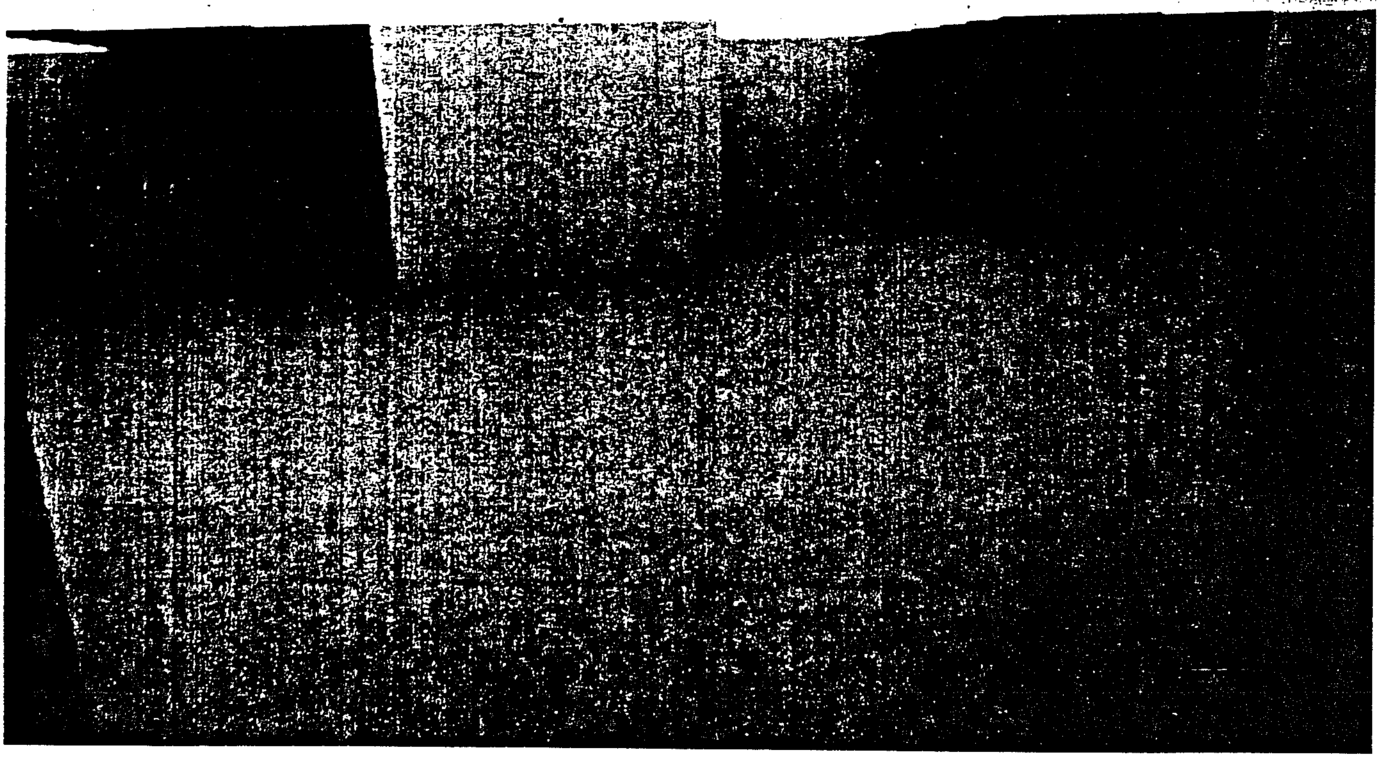
values are luminances in candela/m2



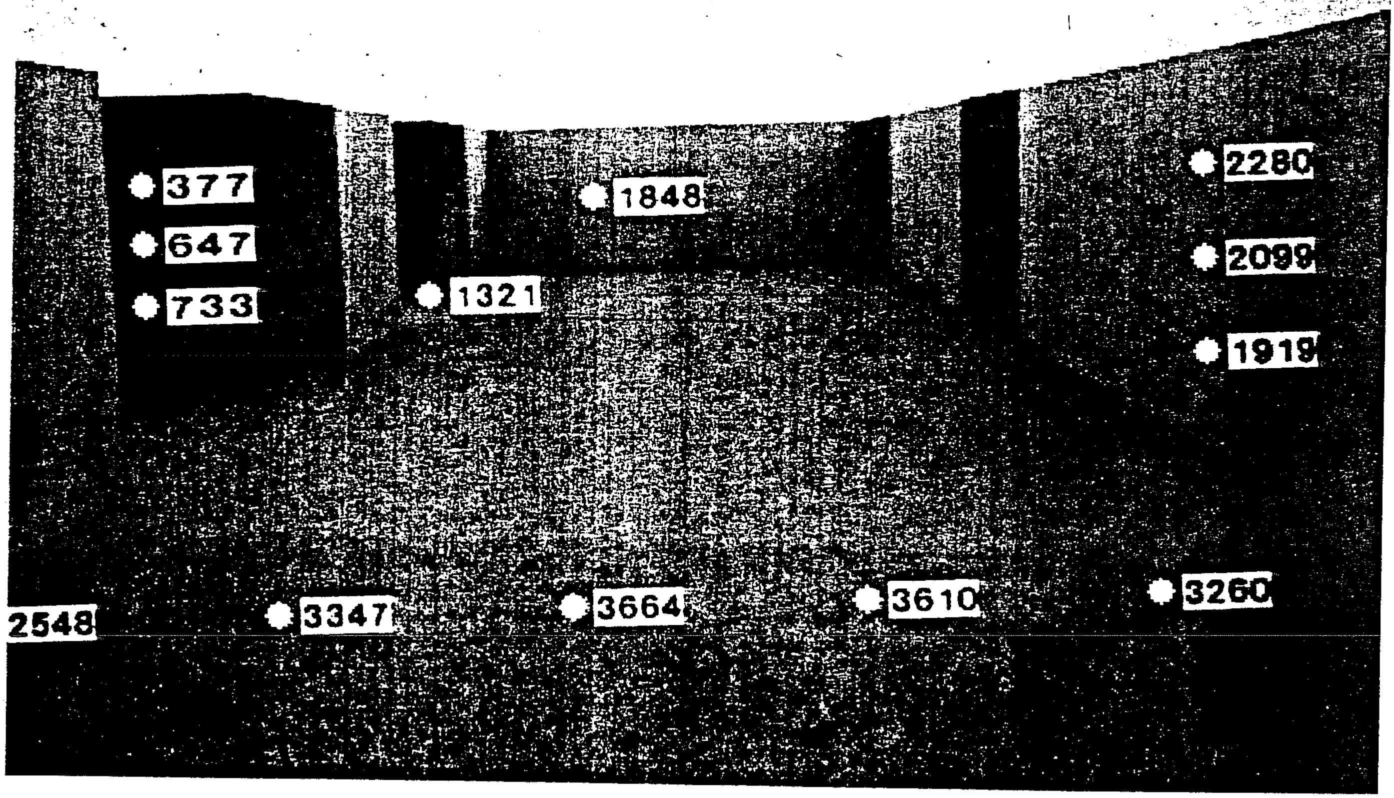


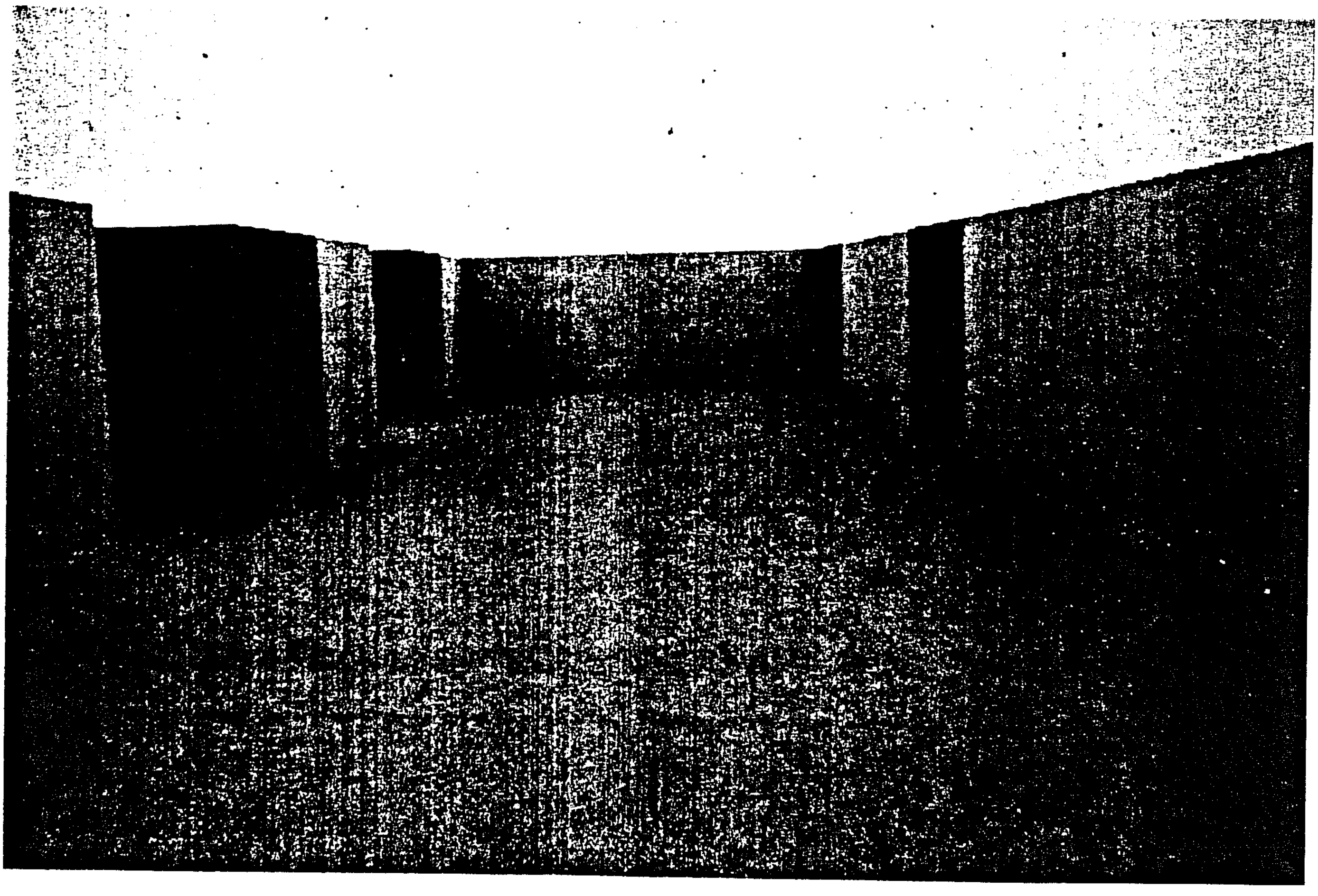
lower.fin.500x338



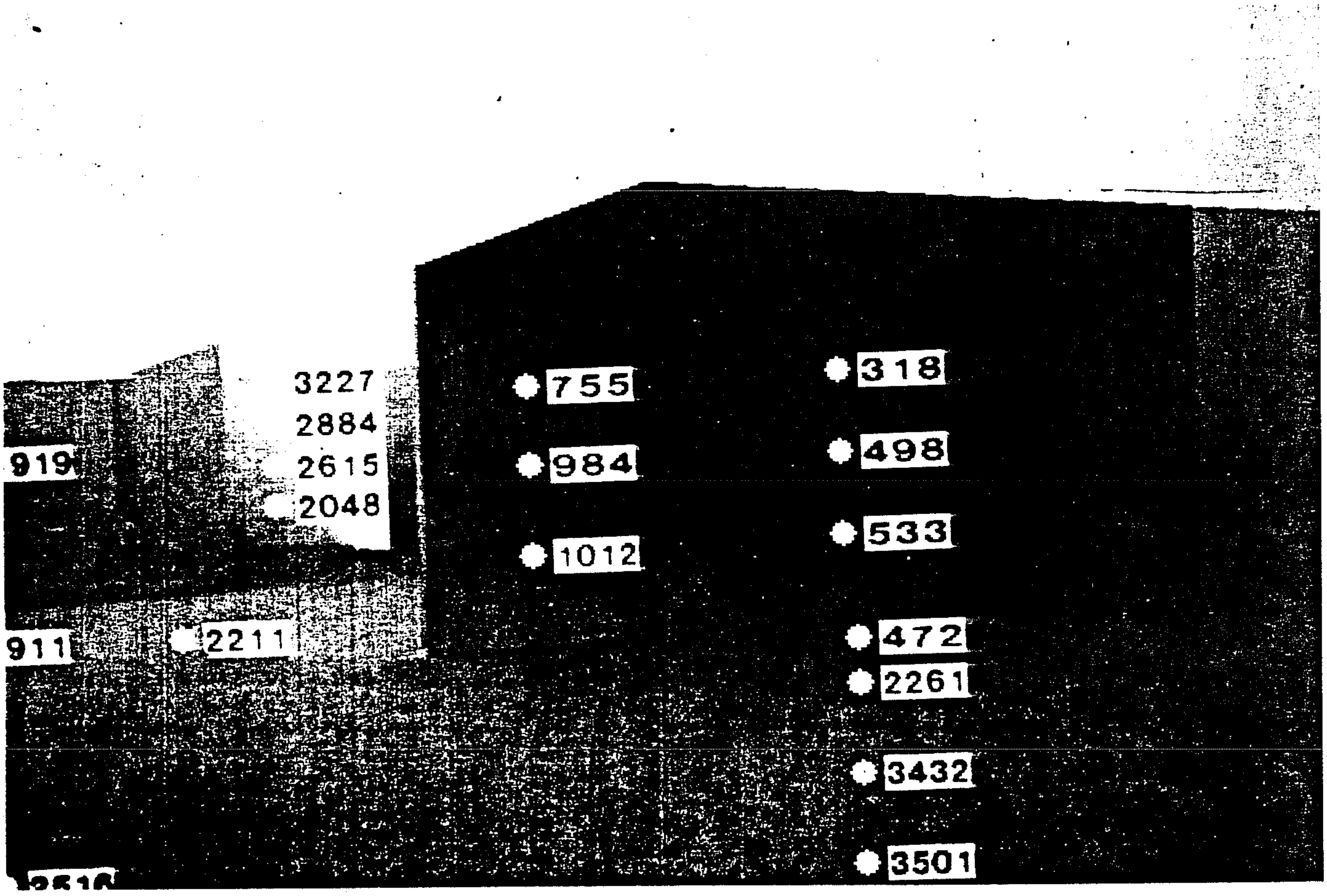


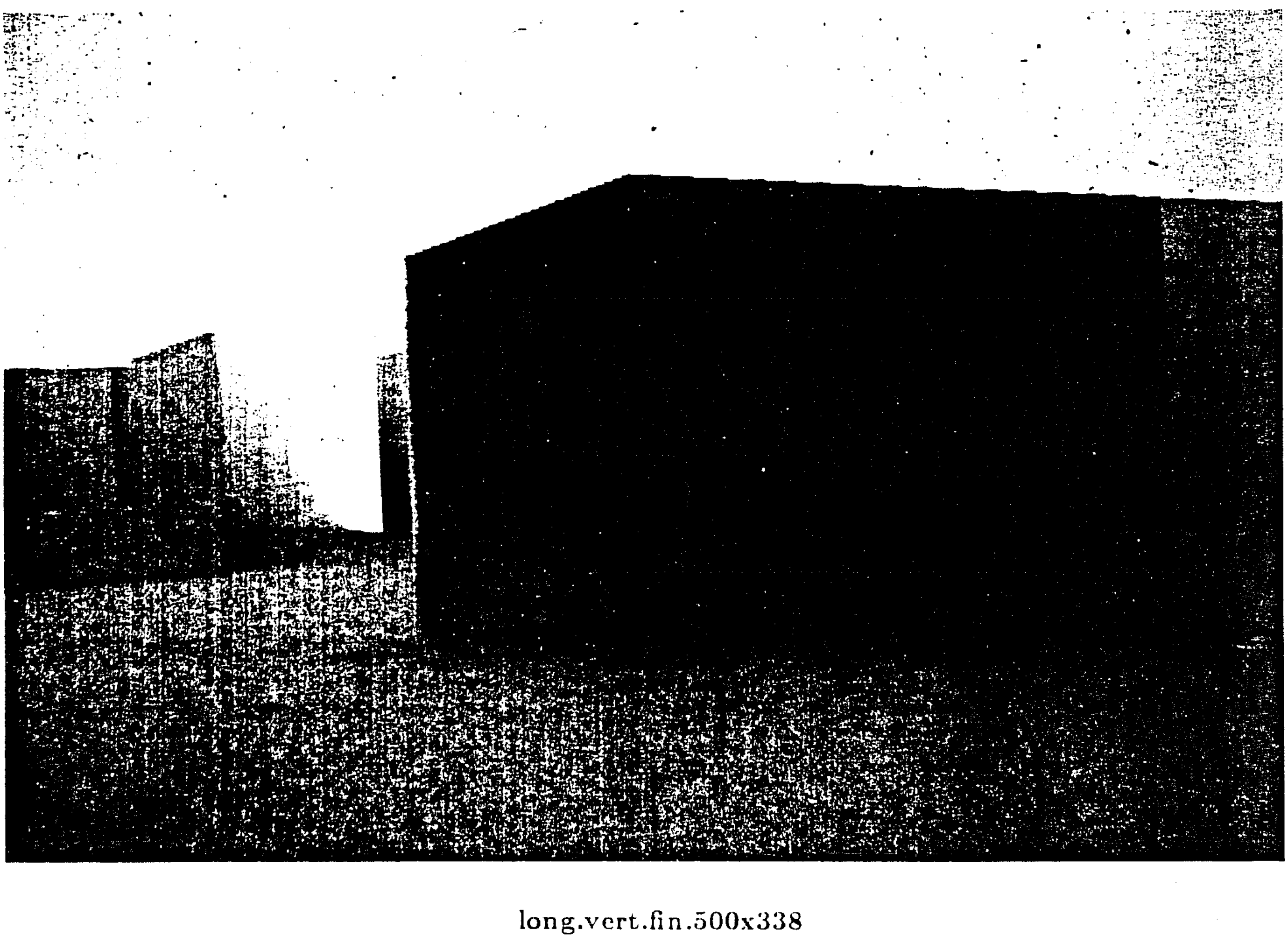
alcove2n3.fin.500x338





long.fin.500x338





long.vert.fin.500x338

CONCLUSION

This work covers a large range of subjects related to the Radiance lighting simulation. An important issue was to demonstrate the use of Radiance for lighting design, so we created the example detailed in the paper of the Vancouver Building Simulation conference.

Since the quantitative validation conducted earlier was successful, we felt that a qualitative validation modeling an existing space would help to convince users of the program's trustworthiness. The results convinced us, and all the other people we showed it to, that Radiance is capable of modeling complex environments. Nevertheless, modeling such a complicated space is still a painful process, and new tools such as CAD systems should be combined in order to make the system easy to use.

With any analysis tool, case studies are needed to demonstrate the program in actual use. The Museum represents a case study where Radiance was used to analyze a daylighted museum design. The work should be continued for different times of day, year, and different materials.

Finally, we should say that, as a lighting simulation, Radiance has a lot of potential. Some work is required, though, to build material and object libraries that would make choices of elements easier and faster, and to improve the "user-friendliness" of the program.

Acknowledgements

I would like to thank here Greg Ward for his help and support during the time of this work, Steve Selkowitz and Sam Berman who made my work in this nice and stimulating environment possible during my internship concluding my engineering studies at the Formation d'Ingenieurs en Sciences et Technologies, University of Paris 6, France. This work was supported by the Assistant Secretary for Conservation and Renewable Energy, Office of Building Energy Research and Development, Buildings Equipment Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Terminology

Luminance (in a given direction, at a point on the surface of a source or a receptor, or at a point on the path of a beam)

Quotient of the luminous flux leaving, arriving at or passing through an element of surface at this point and propagated in directions defined by an elementary cone containing the given direction, by the product of the solid angle of the cone and the area of the orthogonal projection of the element of surface on a plane perpendicular to the given direction.

Symbols: L_v, L. $\quad L_v = \dfrac{d^2\Phi_v}{d\Omega \cdot dA \cdot \cos\theta}$.

Unit: candela per square meter $cd \cdot m^{-2}$

Illuminance; illumination (at a point of a surface)

Quotient of the luminous flux incident on an element of the surface containing the point, by the area of that element.

Symbols: E_v, E, $\quad E_v = \dfrac{d\Phi_v}{dA}$

Units: lux, lx

Reflectance:

Ratio of the reflected radiant or luminous flux to the incident flux.

Symbols: ρ_e, ρ_v, ρ $\quad \rho=\rho_r+\rho_d$

Note 1. Where mixed reflection occurs, the (total) reflectance may be divided into two parts, **regular reflectance** (ρ_r) and **diffuse reflectance** (ρ_d).

Radiant emittance:

Most incandescent sources have well-defined radiating surfaces. The radiant emittance of an area on such a source is defined as the flux emitted by that area divided by the area.

Average radiant emittance is the total source radiant flux divided by the total radiating area. Radiant emittance is defined more generally as a point function by taking the limiting case

$$M_x = dP/dA]_x$$

Where M is the radiant emittance at point x, P is radiant flux and A is area.
Unit: M is in Watt/m^2, P in Watt and A in Meters.

Irradiance on the surface is the incident flux per unit area.
The general definition is

$$E_x = dP/dA]_x$$

where E is the irradiance on point x, P the incident radiant flux and A the area. Unit: E is in Watt/m^2, P in Watt and A in Meters.

Radiance represents the same quantity as luminance, but the units are different Radiance is in $Watts/sr/m^2$ whereas the luminance is in $Candela/m^2$. The reason of this is that radiance is a radiometric unit (in terms of power) and luminance is a photometric unit (in terms of a light response of the eye).

The following table represents a summary of the general parameters used in lighting.

PARAMETER SUMMARY

	Radiant System		**Luminous System**	
Standard Symbol	**Name**	**SI Units**	**Name**	**SI Units**
Q	Radiant energy	*Joule*	Luminous energy	*Talbot*
Φ	Radiant flux	*Watt*	Luminous flux	*Lumen*
M	Radiant emittance	$Watt/m^2$	Luminous emittance	$lumen/m^2$
I	Radiant intensity	$Watt/sr$	Luminous intensity	*Candela*
L	Radiance	$Watt/m^2 \cdot sr$	Luminance	$Candela/m^2$
E	Irradiance	$Watt/m^2$	Illuminance	$Lumen/m^2$

References

[Kaufman81] Kaufman, J.E., *IES Lighting Handbook,* (Reference and Application Volumes), Illuminating Engineering Society of North America, 1981.

[Siminovitch87] Siminovitch, Navvab and Rubinstein, "The Effects of Interior Room Cavity Obstruction on the Illuminance Distribution Characteristics in Task Station Application", *Proceedings of the annual IEEE Industry Applications Society Conference,* Atlanta Georgia, October 1987.

[Ballman87] Ballman, T.L. and R.E. Levin, "Illumination in Partitioned Spaces", *Journal of the IES,* Vol 16. No. 2, 1987, pp. 31-49.

[Grynberg88] A. Grynberg Comparison and Validation of Radiance and Superlite, internal report.

[Ward88] G. J. Ward and F. M. Rubinstein, "A New Technique for Computer Simulation of Illuminated Spaces", *JOURNAL of the Illuninating Engineering Society,* Winter 1988.

[Ward89] "The RADIANCE Synthetic Imaging System" *Reference Manuel for the Radiance programme* Greg Ward version 1.0 january 1989.

[Ward89.6] G. Ward, F. Rubinstein, A. Grynberg *Luminance in coumputer-Aided Lighting Design* Proceedings of Building Simulation '89 June 23 - 24, 1989.

[NCSA] ImageTool for the Sun Workstation Version 1.0 *ImageTool V1.0 manual.* May 1988.
Authors : Dr. Michael L. Norman and Carol Song.
NCSA 152 Computing Applications Building
605 East Springfield Ave.
Champaign, Illinois 61820
ImageTool for the Sun Workstation Version 1.0

[Bevington] Philip R. Bevington, "Data Reduction and Error Analysis for the Physical Sciences", *McGraw-Hill Book Compagny* 1969.

Appendix 1

Example of a Makefile

```
#
# Makefile for the conference room
#

VIEW = -vf vf/current
SCENE = electric
DEV = X
OCTOPTS = -f

view: $(SCENE).oct
	rview $(VIEW) -o $(DEV) -av .02 .02 .02 $(SCENE).oct

electric.oct:furnished.oct lights.fi lights.fo lights.ri fixture curtains curtain.oct
	oconv $(OCTOPTS) -i furnished.oct curtains lights.fi lights.fo lights.ri lights.ro > electric.oc

test.oct:	furnished.oct test_light
	oconv $(OCTOPTS) -i furnished.oct test_light > test.oct

furnished.oct:	conf.oct furniture table table.pts wastebasket waste.cal podium chair1.oct chair2.
	oconv $(OCTOPTS) -i conf.oct furniture > furnished.oct

conf.oct:	conf door_jam.pts screws screw window1 window2 blackboard small_board pinboard cc
	oconv $(OCTOPTS) -b -2 -9 -15 40 conf > conf.oct

chair1.oct: chair1 chair1back.pts chair1angles.pts
	oconv -f chair1 > chair1.oct

chair2.oct: chair2 chair2back.pts chair2leg.pts chair2arm.pts chair2seat.pts
	oconv -f chair2 > chair2.oct

curtain.oct:	curtain
	oconv -f curtain > curtain.oct

vent.oct:	vent vent.fmt
	oconv -f vent > vent.oct
```

Explanations

We put the view parameters of a picture (view point, view direction, view angle, exposure,...) in files and in order to differentiate them easily to the other files, and transferred all this files in a subdirectory called *vf*.

VIEW, SCENE, DEV, OCTOPTS are parameters. One can change them without having to change the all Makefile.

- **VIEW** set the view file.
- **SCENE** is the name of the version of the picture. Electric stand for electric light. Another version with daylight should be done.
- **DEV** is the name of the output device for rview. X = Xwindow.
- **OCTOPTS** contains the options to oconv, so the same kind of octree is done uniformly.
- Files ending with **.oct** are the octree files.
- Files endig with **.pts** are files containing points values or coordinates.
- file ending with **.cal** are calculation file for pattern or textures.
- File ending with **.fmt** are format file. For example, "vent.fmt" contain the parametric description for a vent. "vent.fmt" is called by a "rcalc" inside of "vent" to make the automatic generation of a panel of vents.

Appendix 2

Example of a Radiance fixture file.

```
# xform -rz 90 -t 24 12 106.7
#
# Wellmade 2 lamp fixture, model 325-AL97-248RS
#
# 2x4' flush mount fluorescent with .5" metalized parabolic louver
#
# Dimensions in inches, fixture parallel to y axis, center at origin
#

void brightdata lum_dist
11 flatcorr source/wellmade/325-AL97-248RS.dat source.cal src_phi4 src_theta -rz 90 -t 24 12 106.7
0
0
# this is scaled by 1/area (SI)

lum_dist light lum_bright
0
0
3          1.55          1.55          1.55

lum_bright polygon luminaire
0
0
12
        46.75          1.25          106.7
         1.25          1.25          106.7
         1.25         22.75          106.7
        46.75         22.75          106.7
```

Appendix 3

Fixture data file

All the fixtures and lamp data are contained in /usr/local/lib/ray/source and then, under manufacturer name. This particular one is made by **Wellmade.**

Attention, in a normal data file, no comments are allowed.

The 3 first lines explain the file format. They are 2 variable Φ and Θ.

Φ varys from 0 to 90 degrees and takes 3 values,
Θ varys from 0 to 90 degrees and takes 11 values.

The given values are the candlepower distribution (candelas or cd) converted into watts/steradian. The convertion of candlepower to watts/steradian is:

$$\frac{candelas}{683}$$

where 683 is the lumens/watt for a monochromatic source. The final data for Radiance are given according to the order set in the format:

$\phi = 0$	and	$\Theta = 0$
		$\Theta=10$
		$\Theta = 20$
		..
$\Phi = 45$	and	$\Theta = 0$
..		..

The data can be given in row or column, the importance is to keep the order of the angles Θ and Φ consistant. The following is the effective file for Radiance

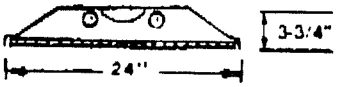
325-AL97-248RS.dat.

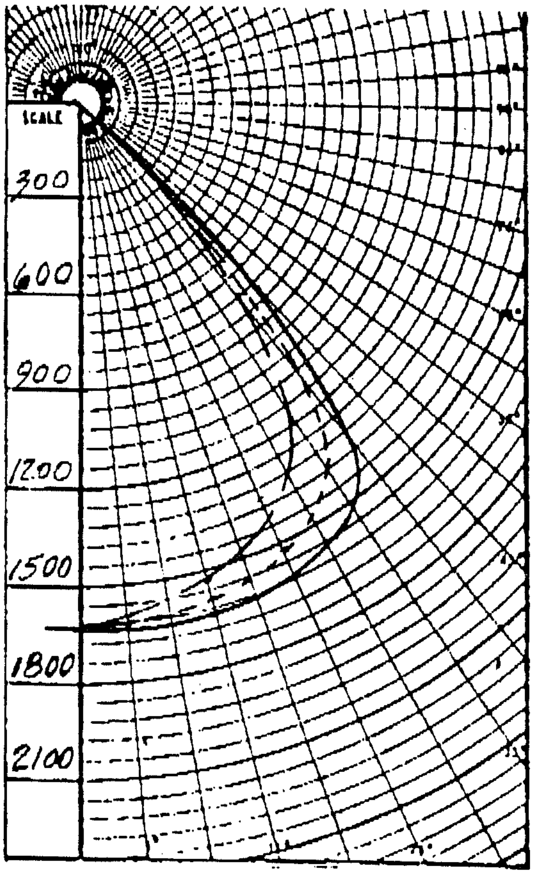
```
2
0 90 3
0 90 11

2.38 2.34 2.23 2.00 1.70 0.78 0.02 0.01 0.01 0.01 0
2.38 2.38 2.34 2.21 2.02 1.21 0.03 0.01 0.01 0.01 0
2.38 2.39 2.39 2.36 2.26 1.15 0.02 0.02 0.01 0.01 0
```

Appendix 4

WELLMADE METAL PRODUCTS COMPANY
REPORT PREPARED FOR: WELLMADE METAL PRODUCTS CO.
CATALOG NUMBER: 325-AL97-248RS

REPORT NO: 442 LAMPS: 2 F40CW LUMENS: 3150
IES SPACING CRITERION: 1.2 CU'S BASED ON S/MH RATIO= .4
DESCRIP.: 2 X 4 2-LAMP TROFFER WITH AL97 LOUVER





ZONAL SUMMARY

ZONE DEG.	AVG* C.P.	ZONA LUMEN
180	0	
175	0	0
165	0	0
155	0	0
145	0	0
135	0	0
125	0	0
115	0	0
105	0	0
95	0	0
90	0	0
85	8	8
75	9	9
65	11	11
55	22	20
45	746	578
35	1369	860
25	1504	696
15	1590	451
5	1623	155
0	1628	

CEILING CAVITY REFLECTANCE

WALL REFLECTANCE

COEFFICIENTS OF UTILIZATION FOR FL. CAV. REFL.= 20 %

CEILING CAVITY REFLECTANCE	80				70				50			30			10			0
WALL REFLECTANCE / ROOM CAVITY RATIO	70	50	30	10	70	50	30	10	50	30	10	50	30	10	50	30	10	0
0	53	53	53	53	51	51	51	51	49	49	49	47	47	47	45	45	45	44
1	50	49	47	46	49	48	47	46	46	45	44	44	44	43	43	42	42	41
2	47	45	43	41	46	44	42	41	43	41	40	41	40	39	40	39	38	38
3	45	41	39	37	44	41	39	37	40	38	36	39	37	36	37	36	35	34
4	42	38	35	33	41	38	35	33	37	34	33	36	34	32	35	33	32	31
5	39	35	32	30	38	35	32	30	34	31	30	33	31	29	32	31	29	28
6	37	32	30	27	36	32	29	27	31	29	27	31	29	27	30	28	27	26
7	34	30	27	25	34	29	27	25	29	26	24	28	26	24	28	26	24	23
8	32	27	24	22	31	27	24	22	26	24	22	26	23	22	25	23	22	21
9	30	25	21	19	29	24	21	19	24	21	19	23	21	19	23	21	19	18
10	28	22	19	17	27	22	19	17	22	19	17	21	19	17	21	19	17	16

THESE COEFFICIENTS WERE COMPUTED BY THE ZONAL-CAVITY METHOD, I.E.S. RECOMMENDED PRACTICE, AND PREPARED FROM THE CANDLEPOWER DISTRIBUTION DATA IN PHOTOMETRIC REPORT 442

* AVERAGE OVER ZONE. FOR SPECIFIC CANDELA VALUES, SEE P. 2.

WELLMADE METAL PRODUCTS CO. 325-AL97-248RS

CANDLEPOWER

ZONE DEG.	0	45	90
		CANDELAS	
90	0	0	0
85	7	7	8
75	8	8	9
65	9	10	14
55	19	25	19
45	539	828	791
35	1163	1386	1544
25	1367	1516	1618
15	1525	1600	1637
5	1600	1628	1637
0	1628	1628	1628

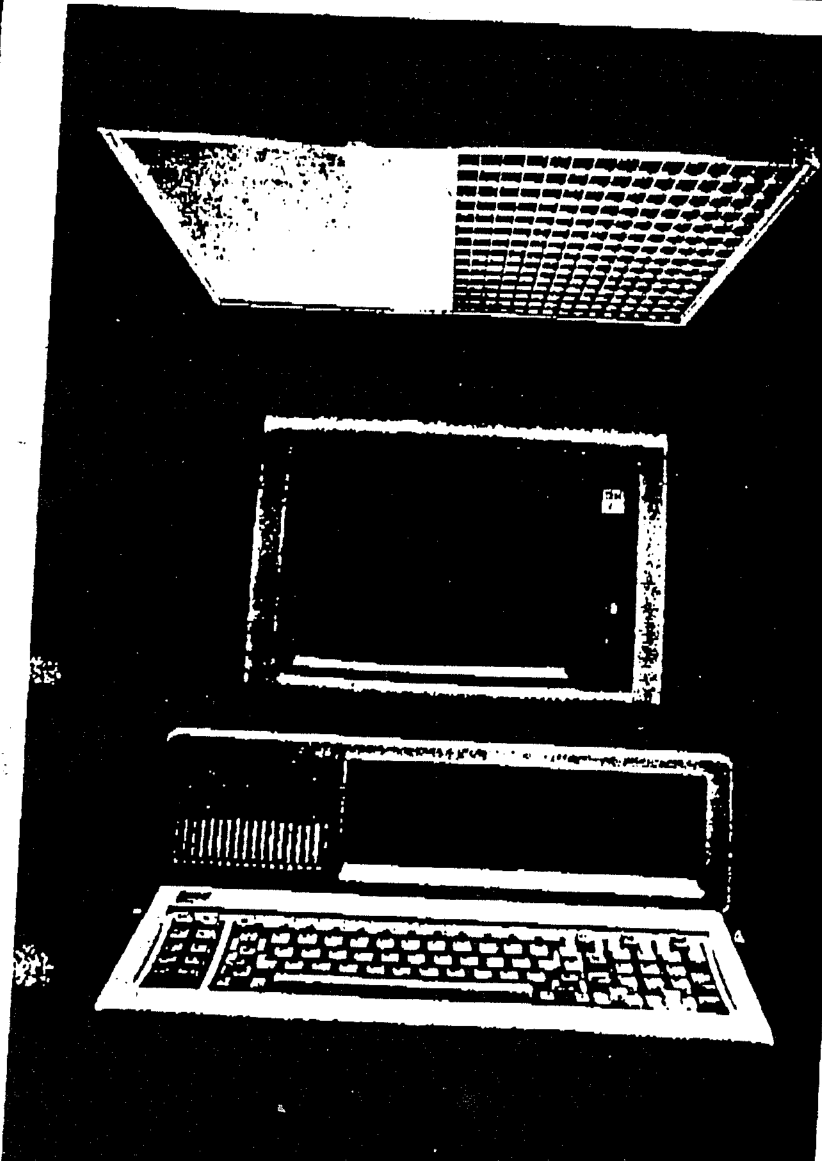
ZONE	LUMENS	% BARELAMP	% LUMINAIRE
0 TO 30	1302	20.7	46.7
0 TO 40	2162	34.3	77.6
0 TO 60	2760	43.8	99.0
0 TO 90	2788	44.3	100.0
90 TO 180	0	0.0	0.0
0 TO 180	2788	44.3	100.0

LUMINANCE VALUES

ANG.	AVG. DIAG	AVG. PERP
85	40	44
75	15	17
65	11	15
55	20	15
45	542	518

screen were found!

LIGHT YEARS AHEAD with PARACUBE I H.E.F. LOUVERS

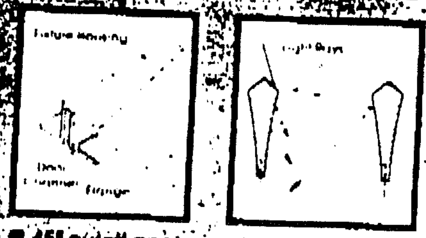


The compatibility of fluorescent lighting, small cube parabolic louvers and CRT screens is generally acknowledged. American Louver Company now introduces new features that put the PARACUBE I H.E.F. (High Efficiency Flange) ½" x ½" louver* ahead of the ... "light years" ahead.

New! ■ Domed specular ... parabolic blades redirect ... downward. The result? PARACUBE I H.E.F. louvers are now the most efficient ½" cell louvers in the industry, 35% more efficient than even our original PARACUBE I louver.

New! ■ An integral ... flange provides uniform brightness control across the entire louver, including perimeter area.

New! ■ Two flange widths will fit virtually any 2x4 fluorescent fixture door frame.



■ 45° cutoff angle results in glare-free ...

Specify the best... Paracube I H.E.F. louvers from

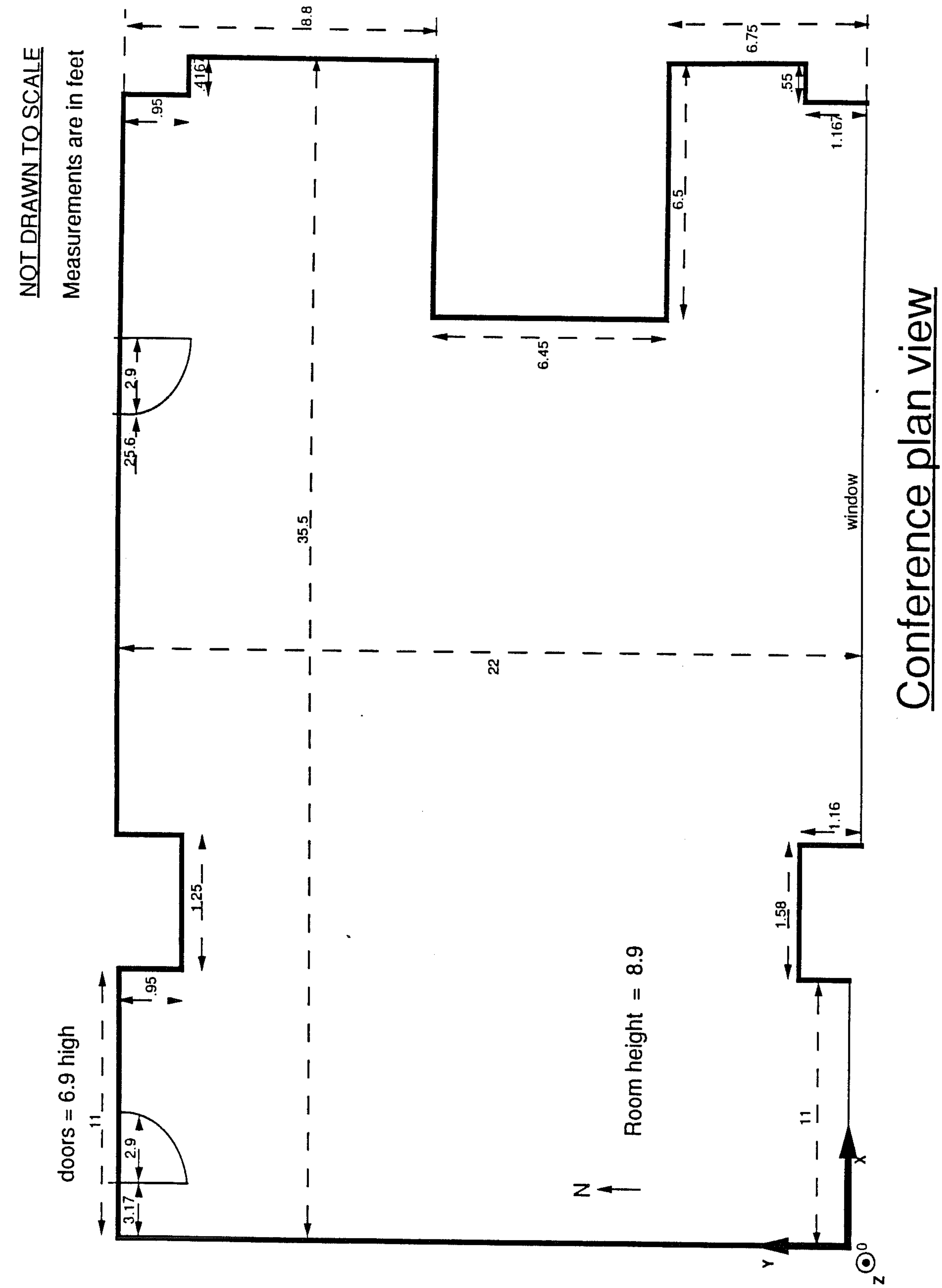
AMERICAN LOUVER COMPANY

For literature or more information call 1 800-323-4250

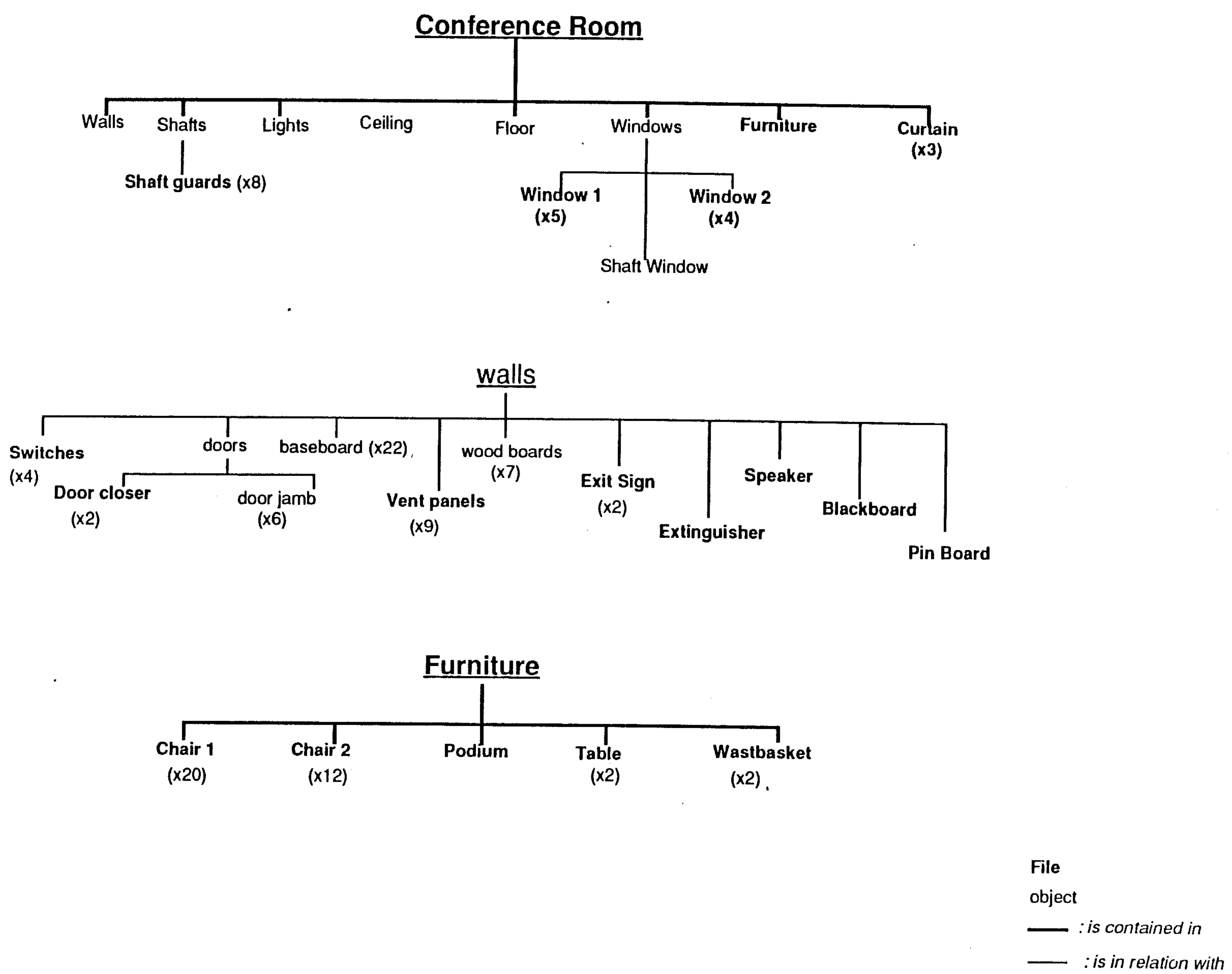
7700 N. AUSTIN AVENUE • SKOKIE, IL 60077 • IN ILLINOIS (312) 470-3300

Circle 18

Appendix 5

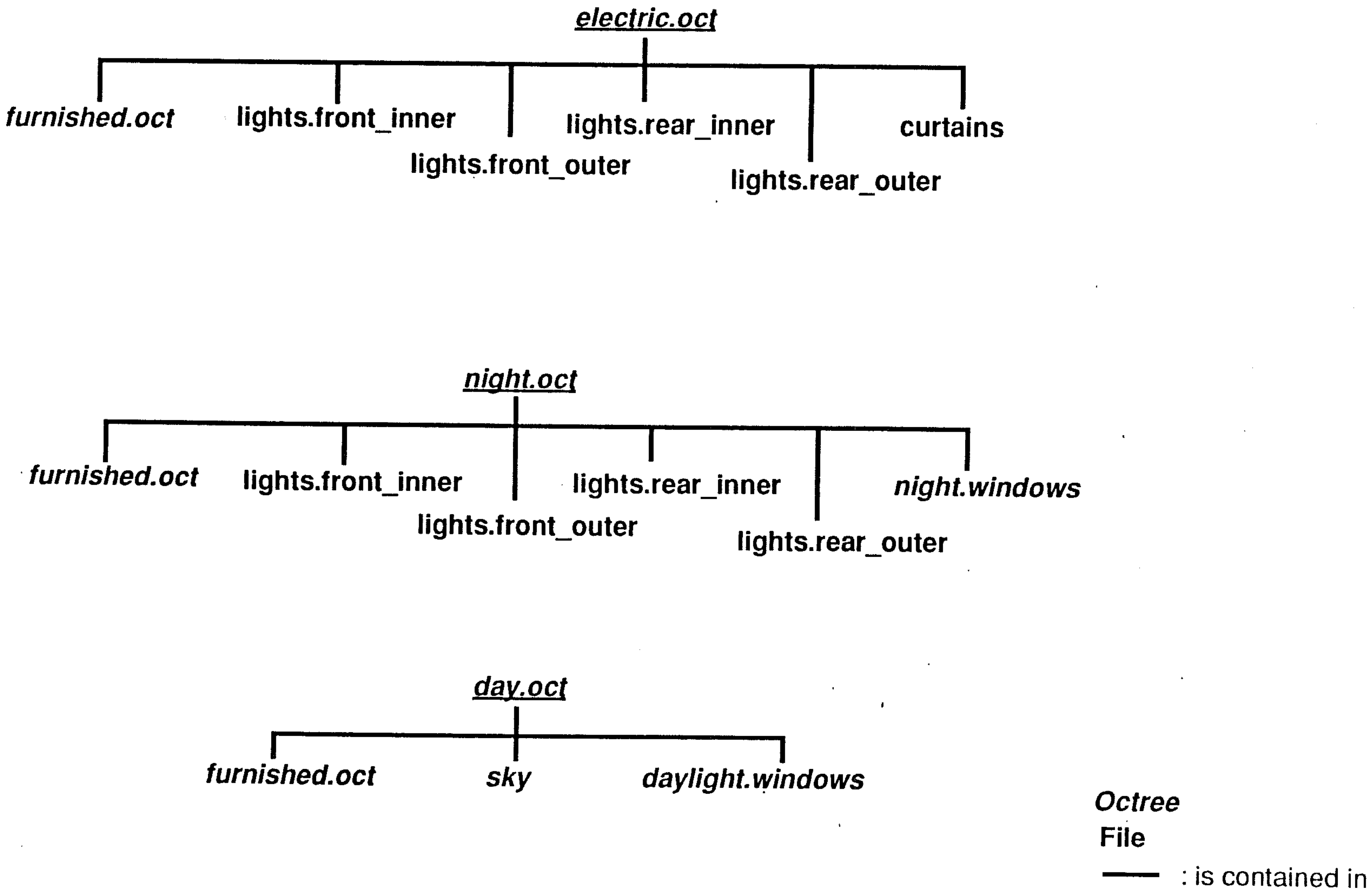


NOT DRAWN TO SCALE
Measurements are in feet
Conference plan view
doors = 6.9 high
Room height = 8.9
window
N
X
Y
Z
8.8
6.75
.416
.95
.55
1.167
6.5
6.45
2.9
25.6
35.5
22
1.16
1.25
1.58
.95
11
2.9
3.17
11



Conference Room
Walls
Shafts
Lights
Ceiling
Floor
Windows
Furniture
Curtain (x3)
Shaft guards (x8)
Window 1 (x5)
Window 2 (x4)
Shaft Window
walls
Switches (x4)
doors
baseboard (x22)
wood boards (x7)
Exit Sign (x2)
Speaker
Blackboard
Pin Board
Extinguisher
Vent panels (x9)
Door closer (x2)
door jamb (x6)
Furniture
Chair 1 (x20)
Chair 2 (x12)
Podium
Table (x2)
Wastbasket (x2)
File
object
: is contained in
: is in relation with

Hierarchy in the Octrees



Files dependencies

Conf.oct

door_jam
screws screw
window1.frame window2.frame
blackboard small_board pinboard
shaft_guard
door closer
exit_sign
plate
speaker
extinguisher
vent_panel vent.oct

Furnished.oct

conf.oct furniture
table table.pts table.stand
wastebasket
podium
chair1.oct chair2.oct

Chair1.oct

chair1
chair1back.pts chair1angles.pts

File

dependencies

Appendix 6

The following are the measurements of the radius of the distorted sphere. Phi and Theta are in degrees. The Radius is in millimeters.

Phi	*Theta*	*Radius*
0	0	245
0	15	235
0	30	223
0	45	212
0	60	205
0	75	201
0	90	200
0	105	200
0	120	202
0	135	205
0	150	211
0	165	220
0	180	228

Phi	*Theta*	*Radius*
30	0	247
30	15	238
30	30	227
30	45	217
30	60	209
30	75	204
30	90	200
30	105	202
30	120	203
30	135	206
30	150	211
30	165	220
30	180	230

Phi	*Theta*	*Radius*
60	0	245
60	15	238
60	30	237
60	45	220
60	60	210
60	75	204
60	90	200
60	105	199
60	120	200
60	135	204
60	150	210
60	165	220
60	180	233

Phi	*Theta*	*Radius*
90	0	240
90	15	233
90	30	225
90	45	216
90	60	210
90	75	202
90	90	200
90	105	198
90	120	202
90	135	205
90	150	215
90	165	226
90	180	238

Phi	*Theta*	*Radius*
120	0	234
120	15	236
120	30	218
120	45	210
120	60	205
120	75	202
120	90	200
120	105	198
120	120	202
120	135	209
120	150	218
120	165	232
120	180	244

Phi	*Theta*	*Radius*
150	0	235
150	15	225
150	30	216
150	45	209
150	60	205
150	75	200
150	90	200
150	105	200
150	120	202
150	135	209
150	150	220
150	165	233
150	180	244

Phi	*Theta*	*Radius*
180	0	234
180	15	223
180	30	215
180	45	209
180	60	205
180	75	203
180	90	202
180	105	202
180	120	205
180	135	213
180	150	223
180	165	235
180	180	245

Appendix 7 (1)

```
/*
 * the aim of this program is to define numericaly the center of the sphere
 *
 * 3/15/89
 */
#include <stdio.h>
#define PI 3.14159265358979323846
#define d2r(d) ((d) * (PI/180)) /* convert degrees into radians */
#define sqr(d) ((d) * (d))

double x[100],y[100],z[100];
int n;

double sin() , cos() ;
double fabs(), sqrt();

acquisition()
/* this function stores the data */

{
double theta,phi,r;

n=0;
while(scanf("%lf %lf %lf",&theta,&phi,&r)==3)
   {
   x[n]=r * cos(d2r(theta)) * cos(d2r(phi));
   y[n]=r * sin(d2r(theta)) * cos(d2r(phi));
   z[n]=r * sin(d2r(phi));
   n++;
   }
}; /* end of acquisition */

double
chi_sqr(X,Y,Z,R)
/* this function calculates chi square */

double X,Y,Z,R;
{
double sum, d;
register int j;

sum=0;
for(j=0;j<n;j++) {
   d=sqrt(sqr(x[j]-X) + sqr(y[j]-Y) + sqr(z[j]-Z)) - R ;
   sum += sqr(d);
}
return(sum);
};     /* end of chi*/

points(X1,X2,X3,Xmin)
double *X1, *X2, *X3, *Xmin;
{
if( *X2 < *Xmin && *Xmin < *X3)
   {
   *X1 = *X2;
   *X2 = *Xmin;
/* *X3 = *X3;*/
   }
else
   if( *X1 < *Xmin && *Xmin < *X2)
      {
   /* *X1 = *X1;*/
      *X3 = *X2;
      *X2 = *Xmin;
      }
```

acquisition

chi_sqr

points

```c
   else
      if( *Xmin  >  *X3)
         {
         *X1 = *X2;
         *X2 = *X3;
         *X3 = *Xmin;
         }
      else
         if( *Xmin  <  *X1)
            {
            *X3 = *X2;
            *X2 = *X1;
            *X1 = *Xmin;
            }

 } /* end of points */

 double atof();

 double minx(x1,y1,x2,y2,x3,y3)
 double x1,y1,x2,y2,x3,y3;
 {
                                                          /* second derivative */
  if ( ((y3-y2)/(x3-x2) - (y2-y1)/(x2-x1)) / (x3-x1) <= 0. )
          return( y3<y1 ? 2.*x3-x2 : 2.*x1-x2 );             /* assume x1<x2<x3 */

  return( .5 * ( -(y2-y1)/(x2-x1) * (x3-x1)/((y3-y2)/(x3-x2)-(y2-y1)/(x2-x1))+(x2+x1)) );
 } /* end of minx */
```

Min

```c
 Min(X,Y,Z,R)
 double X,Y,Z,R;
 {
 #define del  10
 #define eps  .05

 double X1,X2,X3,Xmin;
 double Y1,Y2,Y3,Ymin;
 double Z1,Z2,Z3,Zmin;
 double R1,R2,R3,Rmin;


 X1=X-del;
 X2=X;
 X3=X+del;

 Y1=Y-del;
 Y2=Y;
 Y3=Y+del;

 Z1=Z-del;
 Z2=Z;
 Z3=Z+del;

 R1=R-del;
 R2=R;
 R3=R+del;


 do{
    X=X2; Y=Y2; Z=Z2; R=R2;

    Xmin =
    minx(X1,chi_sqr(X1,Y2,Z2,R2),X2,chi_sqr(X2,Y2,Z2,R2),X3,chi_sqr(X3,Y2,Z2,R2));
```

```
    points(&X1,&X2,&X3,&Xmin);

    Ymin =
    minx(Y1,chi_sqr(X2,Y1,Z2,R2),Y2,chi_sqr(X2,Y2,Z2,R2),Y3,chi_sqr(X2,Y3,Z2,R2));

    points(&Y1,&Y2,&Y3,&Ymin);

    Zmin =
    minx(Z1,chi_sqr(X2,Y2,Z1,R2),Z2,chi_sqr(X2,Y2,Z2,R2),Z3,chi_sqr(X2,Y2,Z3,R2));

    points(&Z1,&Z2,&Z3,&Zmin);

    Rmin =
    minx(R1,chi_sqr(X2,Y2,Z2,R1),R2,chi_sqr(X2,Y2,Z2,R2),R3,chi_sqr(X2,Y2,Z2,R3));

    points(&R1,&R2,&R3,&Rmin);
    }

 while ( fabs(Xmin-X)>eps || fabs(Ymin-Y)>eps ||fabs(Zmin-Z)>eps ||
          fabs(Rmin-R)>eps );
 /*
 printf(" Xmin = %lf , Ymin = %lf\n",Xmin,Ymin);
 printf(" Zmin = %lf , Rmin = %lf\n",Zmin,Rmin);
 */
 printf("%lf %lf %lf %lf %lf\n",chi_sqr(Xmin,Ymin,Zmin,Rmin),
                         Xmin,Ymin,Zmin,Rmin);

 } /* end of Min */
```



```
main(argc,argv)
int argc;
char *argv[];
{
double X,Y,Z,R;
   if(argc!=5)
      {
      fprintf(stderr,"Usage : %s X0 Y0 Z0 R0\n",argv[0]);
      exit(1);
      };
X=atof(argv[1]); /* convert a string into a number */
Y=atof(argv[2]);
Z=atof(argv[3]);
R=atof(argv[4]);
acquisition();
Min(X,Y,Z,R);
} /* end of main */
```

Appendix 8 (1)

```
#include <math.h>
#include <stdio.h>

#define PI 3.14592653589793238 46
#define d2r(d) ( (d) * (PI / 180 ) ) /* convert degree into radians */
#define sqr(d) ((d)*(d))

typedef struct {double x, y;}point, *ppoint;

double sin() , cos() ;
double fabs(), sqrt() ;

double tab[91][3];
int n;
```

circumcircle

```
circumcircle(p, q, r, vect)
/*
* circumcircle: find circle thru 3 points p, q, r
* puts center in cen during the program. At the end puts the
* normalized normal vector in vect, returns radius
* p has to be the curent point.
*/
point *p, *q, *r, *vect;
{
     double a1, b1, c1, a2, b2, c2, den,quot,w,z;
     point cen;

     a1 = (q->x - p->x);
     b1 = (q->y-p->y);
     c1 = -.5*(a1*(p->x+q->x)+b1*(p->y+q->y));
     /* perpendicular bisector between p and q is a1*x+b1*y+c1=0 */
     a2 = r->x-q->x;
     b2 = r->y-q->y;
     c2 = -.5*(a2*(q->x+r->x)+b2*(q->y+r->y));
     /* perpendicular bisector between q and r is a2*x+b2*y+c2=0 */
     den = a1*b2-a2*b1;
     if (den==0.) {            /* p,q,r are collinear */
        quot = sqrt(sqr(a1) + sqr(b1) );
        vect->x=a1/quot;
        vect->y=b1/quot;
        }
     else {
        cen.x = (b1*c2-b2*c1)/den;
        cen.y = (c1*a2-c2*a1)/den;
        w = p->x - cen.x;
        z = p->y - cen.y;
        quot = sqrt( sqr(w) + sqr(z) );
        vect->x = w/quot;
        vect->y = z/quot;
        }
}
```

acquisition

```
acquisition()
/* this function stores the data */
{
double theta,phi,r;

n=0;
while(scanf("%lf %lf %lf",&theta,&phi,&r)==3)
  {
  tab[n][0] = theta;
  tab[n][1] = phi;
  tab[n][2] = r;
  n++;
```

```
    }
 }; /* end of acquisition */


vnorm_theta(i,j,vect) /* calculates the surface normal when theta is fixed */
int i,j;
ppoint vect;
{
point pt1,pt2,pt_crt;
int k;

k=i*13+j;

if(j==0) /* phi = 0 */
   {
   first_point(vect,1,k);
   return;
   }
if(j==12) /* phi = 180 */
   {
   last_point(vect,1,k);
   return;
   }
pt1.x = tab[k-1][2] * cos( d2r(tab[k-1][1]) ); /* general case */
pt1.y = tab[k-1][2] * sin( d2r(tab[k-1][1]) );
pt2.x = tab[k+1][2] * cos( d2r(tab[k+1][1]) );
pt2.y = tab[k+1][2] * sin( d2r(tab[k+1][1]) );
pt_crt.x = tab[k][2] * cos( d2r(tab[k][1]) );
pt_crt.y = tab[k][2] * sin( d2r(tab[k][1]) );
circumcircle(&pt_crt,&pt1,&pt2,vect);
} /* end of vnorm_theta */
```

vnorm_theta

```
vnorm_phi(i,j,vect) /* calculates the surface normal when phi is fixed */
int i,j;
ppoint vect;
{
point pt1,pt2,pt_crt;
int k;

k=i*13+j;

if(i==0) /* theta = 0 */
   {
   first_point(vect,2,k);
   return;
   }
if(i==6) /* theta = 180 */
   {
   last_point(vect,2,k);
   return;
   }
/* general case */
pt_crt.x = tab[k][2] * cos( d2r(tab[k][0]) );
pt_crt.y = tab[k][2] * sin( d2r(tab[k][0]) );
pt1.x = tab[k-13][2] * cos( d2r(tab[k-13][0]) );
pt1.y = tab[k-13][2] * sin( d2r(tab[k-13][0]) );
pt2.x = tab[k+13][2] * cos( d2r(tab[k+13][0]) );
pt2.y = tab[k+13][2] * sin( d2r(tab[k+13][0]) );
circumcircle(&pt_crt,&pt1,&pt2,vect);
} /* end of vnorm_vect */
```

vnorm_phi

```
first_point(vect,flag,n)
ppoint vect;
int flag,n;
{
```

first_point

```
point  pt1,pt2,pt_crt;
switch(flag){
   case 1 : /* plan (x, z) theta is fixed */
               pt_crt.x = tab[n][2] * cos( d2r(tab[n][1]) ) ;
               pt_crt.y = tab[n][2] * sin( d2r(tab[n][1]) ) ;
               pt1.x = tab[n+1][2] * cos( d2r(tab[n+1][1]) ) ;
               pt1.y = tab[n+1][2] * sin( d2r(tab[n+1][1]) ) ;
               pt2.x = tab[n+2][2] * cos( d2r(tab[n+2][1]) ) ;
               pt2.y = tab[n+2][2] * sin( d2r(tab[n+2][1]) ) ;
               break;
   case 2 : /* plan (x, y) phi is fixed */
               pt_crt.x = tab[n][2] * cos( d2r(tab[n][0]) ) ;
               pt_crt.y = tab[n][2] * sin( d2r(tab[n][0]) ) ;
               pt1.x = tab[n+13][2] * cos( d2r(tab[n+13][0]) );
               pt1.y = tab[n+13][2] * sin( d2r(tab[n+13][0]) );
               pt2.x = tab[n+26][2] * cos( d2r(tab[n+26][0]) );
               pt2.y = tab[n+26][2] * sin( d2r(tab[n+26][0]) );
               break;
   }/* end of the switch */
circumcircle(&pt_crt,&pt1,&pt2,vect);
}/* end of first_point */
```

last_point

```
last_point(vect,flag,n)
ppoint vect;
int flag,n;
{
point  pt1,pt2,pt_crt;
switch(flag){
   case 1 : /* plan (x, z) theta is fixed */
            pt_crt.x = tab[n][2] * cos ( d2r(tab[n][1]) );
            pt_crt.y = tab[n][2] * sin ( d2r(tab[n][1]) );
            pt1.x = tab[n-1][2] * cos( d2r(tab[n-1][1]) ) ;
            pt1.y = tab[n-1][2] * sin( d2r(tab[n-1][1]) ) ;
            pt2.x = tab[n-2][2] * cos( d2r(tab[n-2][1]) ) ;
            pt2.y = tab[n-2][2] * sin( d2r(tab[n-2][1]) ) ;
            break;
   case 2 : /* plan (x, y) phi is fixed */
            pt_crt.x = tab[n][2] * cos ( d2r(tab[n][0]) );
            pt_crt.y = tab[n][2] * sin ( d2r(tab[n][0]) );
            pt1.x = tab[n-13][2] * cos( d2r(tab[n-13][0]) );
            pt1.y = tab[n-13][2] * sin( d2r(tab[n-13][0]) );
            pt2.x = tab[n-26][2] * cos( d2r(tab[n-26][0]) );
            pt2.y = tab[n-26][2] * sin( d2r(tab[n-26][0]) );
            break;
   }/* end of the switch */
circumcircle(&pt_crt,&pt1,&pt2,vect);
}/* end of last_point */
```

main

```
main()
{
int i,j,k;
point vect_theta,vect_phi;

acquisition();

for(i=0;i<7;i++)                  /* theta */
   for(j=0;j<13;j++)                         /* phi */
      {
      k=i*13+j;
      vnorm_theta(i,j,&vect_theta);
      vnorm_phi (i,j,&vect_phi);
      printf("%d     %lf     %lf     %lf\n",k,vect_theta.x * vect_phi.x,
                                            vect_theta.x * vect_phi.y,
                                            vect_theta.y);
```

```
    } /* end of the for */
} /* end of the main */
```